TI 2024-060/III
Tinbergen Institute Discussion Paper

PyTimeVar: A Python Package for Trending Time-Varying Time Series Models

Mingxuan Song[1]
Bernhard van der Sluis[2]
Yicong Lin[3]

1 Vrije Universiteit Amsterdam

2 Erasmus Universiteit Rotterdam

3 Vrije Universiteit Amsterdam and Tinbergen Institute

PyTimeVar: A Python Package for Trending Time-Varying Time Series Models*

Mingxuan Song
Vrije Universiteit Amsterdam

Bernhard van der Sluis
Erasmus Universiteit Rotterdam

Yicong Lin
Vrije Universiteit Amsterdam and Tinbergen Institute

Abstract

Time-varying regression models with trends are commonly used to analyze long-term tendencies and evolving relationships in data. However, statistical inference for parameter paths is challenging, and recent literature has proposed various bootstrap methods to address this issue. Despite this, no software package in any language has yet offered the recently developed tools for conducting inference in time-varying regression models. We propose **PyTimeVar**, a Python package that implements nonparametric estimation along with multiple new bootstrap-assisted inference methods. It provides a range of bootstrap techniques for constructing pointwise confidence intervals and simultaneous bands for parameter curves. Additionally, the package includes four widely used methods for modeling trends and time-varying relationships. This allows users to compare different approaches within a unified environment.

Keywords: time-varying, bootstrap, nonparametric estimation, boosted Hodrick-Prescott filter, power-law trend, score-driven, state-space.

1. Introduction

In this paper, we introduce a Python package that offers state-of-the-art estimation and statistical inference methods for time series regression models with flexible trends and/or time-varying coefficients. These models have gained widespread use in recent empirical research across various fields, including climate and financial studies (see, e.g., Ang and Kristensen

*Corresponding author: Department of Econometrics and Data Science, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV, Amsterdam, the Netherlands. E-mail address: yc.lin@vu.nl.

2012; Guo, Wu, and Yu 2017; Friedrich, Beutner, Reuvers, Smeekes, Urbain, Bader, Franco, Lejeune, and Mahieu 2020a; Kapetanios, Millard, Petrova, and Price 2020; Anand, Paul, and Nair 2023; Ren and Lucey 2023; Umlandt 2023). However, there is a notable lack of integrated packages that support comprehensive analysis within these models. Our proposed Python package, **PyTimeVar**, addresses this gap by offering two key contributions: (1) It introduces a recently developed nonparametric estimation approach, accompanied by various theoretically justified bootstrap-assisted methods for both *pointwise* and *simultaneous* inference (Bühlmann 1998; Zhou and Wu 2010; Friedrich, Smeekes, and Urbain 2020b; Friedrich and Lin 2024; Lin, Song, and van der Sluis 2024). This is important for practitioners, as constructing both pointwise and simultaneous intervals is highly challenging using established asymptotic approximations. While bootstrap methods have been shown to provide accurate empirical coverage, current empirical studies often rely on a naive wild bootstrap, overlooking necessary steps to correct asymptotic bias and account for more complex time series dynamics; see discussions in Friedrich and Lin (2024) and Lin *et al.* (2024), and the references therein. **PyTimeVar** enables users to obtain both confidence intervals and bands, allowing for the evaluation of time variations and assessment of parameter evolution over a specific period using *six* carefully implemented bootstrap methods; (2) It also incorporates *four* alternative, widely-used methods for trend analysis and modeling time-varying relationships: the boosted Hodrick-Prescott (bHP) filter (Phillips and Shi 2021; Biswas, Sabzikar, and Phillips 2024; Mei, Phillips, and Shi 2024), power-law trend models (Robinson 2012; Lin and Reuvers 2024), state-space models (Durbin and Koopman 2012), and score-driven (GAS) models (Harvey 2013; Creal, Koopman, and Lucas 2013), including robust t-GAS models (Harvey and Luati 2014; D'Innocenzo, Luati, and Mazzocchi 2023). This integration allows users to easily compare results across various popular methods tailored to our trending time-varying regression framework, all within a single environment, without adjusting existing packages or switching between programming languages for comparative analysis.

Although no package currently exists that specifically focuses on recent bootstrap-assisted inference methods for time-varying models, some packages in both Python and R offer partial functionalities similar to ours. Below, we discuss these existing packages and their limitations concerning nonparametric time-varying models. In Python, the **KernelReg** class in **statsmodels** supports local linear regressions, similar to those considered here. However, it neither focuses on time-varying models nor provides inference on time-varying coefficient curves. Another Python package, **orbit**, proposed by Ng, Wang, Chen, Yang, and Smyl (2020), includes the kernel-based time-varying regression (**KTR**) class, which allows for the examination of time-varying coefficients. However, this package primarily focuses on Bayesian forecasting methods using state-space models, rather than inference on time-varying coefficients. In R, the **tvReg** package (Casas and Fernandez-Casal 2019) offers several classes of time-varying nonparametric models, making it the closest in focus to ours. However, it only provides pointwise confidence intervals using a naive wild bootstrap method, without considering simultaneous inference or alternative methods for modeling time variations. Additionally, the **tvem** package (Dziak, Coffman, Li, Litson, and Chakraborti 2023) focuses on P-spline-based estimation methods for time-varying coefficient models, but similarly, it only provides point-

wise intervals without incorporating more recent inference techniques.

Moreover, while no package is available for power-law trend analysis to the best of our knowledge, there are numerous references for the other methods we consider, across various environments. Here, we discuss only the most relevant packages to ours. The website https://zhentaoshi.github.io/Boosted_HP_filter/index.html provides comprehensive functionalities of the boosted Hodrick-Prescott filter in Julia, Matlab, Python, R, and Stata. For the Kalman smoother, Python offers implementations through the **statsmodels** (Seabold and Perktold 2010) and **pykalman** (Duckworth and Balatsko 2024) libraries, while R users can utilize the **kalmanfilter** package Hubbard (2024). For score-driven models, an R package is provided by Ardia, Boudt, and Catania (2019). Additional code for score-driven models in various applications is available on the website www.gasmodel.com. The packages mentioned above are potentially of general use. As previously noted, our package unifies these methods for trending time-varying models, as comparative methods of the nonparametric, bootstrap-assisted approach, but does not aim to be an all-encompassing solution.

The package is available on PyPI at https://pypi.org/project/PyTimeVar. This paper uses **PyTimeVar** version 1.0.0 and Python version 3.11.5. The documentation of our package is available at https://pytimevar.readthedocs.io/en/latest/. Finally, the latest (development) version of the **PyTimeVar** package is available on GitHub at https://github.com/bpvand/PyTimeVar. It is important to note that our package relies on several standard Python libraries, including **numpy**, **pandas**, **matplotlib**, **scipy**, **time**, **statsmodels**, **os**, and **tqdm**. These packages are automatically installed when the **PyTimeVar** package is invoked from PyPI, if not already present.

The remainder of this paper is structured as follows. Section 2 introduces the methods included in our package. The detailed implementation of our package is covered in Section 3. Section 4 illustrates the package using a real-life dataset. Section 5 concludes.

2. The methodological framework

We first present the models under consideration (Section 2.1), followed by a discussion of the main estimation and inference framework in our software package (Sections 2.2 - 2.3). Section 2.4 introduces alternative methods available in the **PyTimeVar** package. Our focus is on the practical aspects rather than the technical details. For technical discussions, we refer interested readers to the corresponding references provided in the contexts.

2.1. The set-up

Consider the following time series regression model with a flexible trend and time-varying coefficients:

$$y_t = \beta_{0,t} + \sum_{j=1}^{d} \beta_{j,t} x_{j,t} + u_t = \boldsymbol{\beta}_t^\top \boldsymbol{x}_t + u_t, \qquad t = 1, \ldots, n, \qquad d \geq 0, \tag{1}$$

where y_t represents the dependent variable, $\boldsymbol{x}_t = (1, x_{1,t}, \ldots, x_{d,t})^\top$ is a vector of explanatory

variables, and u_t denotes the error term. Additionally, $\boldsymbol{\beta}_t := (\beta_{0,t}, \beta_{1,t}, \ldots, \beta_{d,t})^\top$ is a $(d+1)$-dimensional vector of coefficients that varies over time to capture dynamic relationships. When $d = 0$, we write $(\boldsymbol{\beta}_t, \boldsymbol{x}_t) = (\beta_{0,t}, 1)$. In this case, Model (1) only contains a deterministic trending component. In practice, we observe the data $\boldsymbol{\mathcal{D}}_n := \{(y_t, \boldsymbol{x}_t),\, t = 1, \ldots, n\}$. Model (1) is extensively used in empirical research due to its interpretability and potential robustness to model misspecification (see, e.g., Ang and Kristensen 2012; Friedrich *et al.* 2020b; Ren and Lucey 2023). The time-dependent parameters $\beta_{j,t}$, $j = 1, \ldots, d$, in Model (1) can be estimated using multiple approaches. Given the limited availability of software packages that implement nonparametric local kernel estimation and recent resampling methods, we primarily focus on these techniques. Alternative approaches are partially covered in Section 2.4 to facilitate convenient comparisons for practitioners.

2.2. Local linear kernel estimation

A popular nonparametric approach assumes that $\boldsymbol{\beta}_t = \boldsymbol{\beta}(\tau_t)$, where $\tau_t = t/n$ and $\boldsymbol{\beta}(\cdot) = (\beta_0(\cdot), \ldots, \beta_d(\cdot))^\top : [0,1] \to \mathbb{R}^{d+1}$ is a vector of smooth, deterministic functions (e.g., Cai 2007; Cai, Li, and Park 2009; Zhou and Wu 2010; Phillips, Li, and Gao 2017; Li, Phillips, and Gao 2020; Nishi 2024). The literature commonly advocates using the local linear kernel method to estimate the unknown vector of functions $\boldsymbol{\beta}(\cdot)$, as it exhibits smaller boundary bias compared to local constant estimation and performs similarly to higher-order local polynomials. For a detailed treatment, see Li and Racine (2007, Chapter 2.4).

Note that the data $\boldsymbol{\mathcal{D}}_n$ are observed discretely at times $t = 1, \ldots, n$. The fundamental idea behind a local (linear) kernel estimator is to use this discrete information to approximate $\boldsymbol{\beta}(\tau)$ for every $\tau \in (0,1)$. Specifically, using a first-order Taylor approximation, for any $\tau \in (0,1)$, we have:

$$\boldsymbol{\beta}(\tau_t) \approx \boldsymbol{\beta}(\tau) + \boldsymbol{\beta}^{(1)}(\tau)(\tau_t - \tau).$$

where $\boldsymbol{\beta}^{(1)}(\cdot) = (\beta_0^{(1)}(\cdot), \ldots, \beta_d^{(1)}(\cdot))^\top$, and $\beta_j^{(1)}(\cdot)$, $j = 0, \ldots, d$, is the first-order derivative of $\beta_j(\cdot)$. Model (1) can then be approximated as

$$y_t \approx \boldsymbol{\beta}(\tau)^\top \boldsymbol{x}_t + \boldsymbol{\beta}^{(1)}(\tau)^\top (\tau_t - \tau)\boldsymbol{x}_t + u_t = \boldsymbol{z}_t(\tau)^\top \boldsymbol{\theta}(\tau) + u_t, \tag{2}$$

where $\boldsymbol{z}_t(\tau) = \left(\boldsymbol{x}_t^\top, \frac{\tau_t - \tau}{h}\boldsymbol{x}_t^\top\right)^\top$ and $\boldsymbol{\theta}(\tau) = \left(\boldsymbol{\beta}(\tau)^\top, h\boldsymbol{\beta}^{(1)}(\tau)^\top\right)^\top$. For $\tau \in (0,1)$, it is intuitive that the data points near $\lfloor n\tau \rfloor$ should be given higher weights compared to those farther from this point when estimating $\boldsymbol{\beta}(\tau)$. Therefore, the local linear estimator (LLE) of $\boldsymbol{\theta}(\tau)$ minimizes the following weighted sum of squares:

$$\hat{\boldsymbol{\theta}}(\tau) = \underset{\boldsymbol{\theta}(\tau)}{\operatorname{argmin}} \sum_{t=1}^{n} \Big(y_t - \boldsymbol{z}_t(\tau)^\top \boldsymbol{\theta}(\tau)\Big)^2 K\left(\frac{\tau_t - \tau}{h}\right), \qquad \forall \tau \in (0,1), \tag{3}$$

where $K(\cdot)$ is a kernel function and $h \downarrow 0$ is a bandwidth. Let $\boldsymbol{Z}(\tau) = (\boldsymbol{z}_1(\tau), \ldots, \boldsymbol{z}_n(\tau))^\top$, $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$, and $\boldsymbol{K}_h(\tau) = \operatorname{diag}\left[K\left(\frac{\tau_1 - \tau}{h}\right), \ldots, K\left(\frac{\tau_n - \tau}{h}\right)\right]$. A closed-form expression of the estimator is then available as follows:

$$\hat{\boldsymbol{\theta}}(\tau) = \begin{pmatrix} \hat{\boldsymbol{\beta}}(\tau) \\ h\hat{\boldsymbol{\beta}}^{(1)}(\tau) \end{pmatrix} = \Big(\boldsymbol{Z}(\tau)\boldsymbol{K}_h(\tau)\boldsymbol{Z}(\tau)^\top\Big)^{-1} \boldsymbol{Z}(\tau)\boldsymbol{K}_h(\tau)\boldsymbol{y}. \tag{4}$$

Implementing (4) requires selecting the kernel function $K(\cdot)$ and the bandwidth parameter $h > 0$. It is well known that nonparametric methods are typically less sensitive to the choice of $K(\cdot)$ and more influenced by the selection of h. A large value of h will produce an overly smooth estimated curve, while a small value will lead to a wiggly estimate. The comprehensive simulation study by Friedrich and Lin (2024) highlights that bandwidth selection has a substantial impact on statistical inference and introduces the data-driven leave-$(2l + 1)$-out local modified-cross-validation (LMCV-l) method, where l is a nonnegative integer. The LMCV method combines the modified cross-validation proposed by Chu and Marron (1991) with the local cross-validation by Vieu (1991) and has consistently performed well across various simulation scenarios, see Lin *et al.* (2024). We provide two alternative bandwidth selection methods: the Akaike information criterion (AIC, Cai 2007) and the generalized cross-validation (GCV, Craven and Wahba 1978) (see Friedrich and Lin 2024, Section 4.1 for details). The local linear estimation, along with a variety of commonly used kernel functions and bandwidth selection methods, is implemented in the '`LocalLinear`' class of the **PyTimeVar** package.

2.3. Bootstrap-assisted inference

To quantify estimation uncertainty in these models, both pointwise confidence intervals and simultaneous bands are useful in empirical studies. Pointwise intervals evaluate whether $\beta_j(\tau)$, $j = 0, \ldots, d$, significantly deviates from zero at a specific time point τ. Simultaneous bands assess whether the coefficient curve differs from zero over a given compact interval G, where $G \subset [0, 1]$. This helps to infer the overall movement of the curve.

Asymptotic approximations of $\hat{\boldsymbol{\beta}}(\cdot)$ under general conditions – potentially allowing for nonlinear dependence structures and nonstationarity – are established in Zhou and Wu (2010) and Lin *et al.* (2024). Therefore, asymptotic methods for constructing confidence intervals and bands are possible. However, as discussed in Friedrich and Lin (2024), asymptotic methods often suffer from slow convergence and depend on nuisance parameters, such as asymptotic bias and long-run variance, which are difficult to estimate in practice and require careful selection of tuning parameters. As a result, multiple *residual-based* bootstrap methods have been developed to address these limitations. When used carefully, residual-based bootstrap methods can automatically replicate asymptotic distributions.

In the **PyTimeVar** package, the function `confidence_bands()` in the '`LocalLinear`' class implements five fixed-design residual-based bootstrap methods: local blockwise wild bootstrap (LBWB), wild bootstrap (WB), sieve bootstrap (SB), sieve wild bootstrap (SWB), and autoregressive wild bootstrap (AWB). Additionally, it includes a multiplier bootstrap (MB) method that necessitates the estimation of nuisance parameters. Unlike residual-based methods, the MB method does not involve resampling from the regression residuals. Since the regressors are not resampled, making it a fixed-design approach, these bootstrap methods generally accommodate quite flexible dynamics of regressors. However, the error process must satisfy certain requirements for each bootstrap scheme to be theoretically valid. With the exception of WB and SB, all methods account for serial dependence and heteroskedas-

Table 1: Comparison of the bootstrap methods for different error processes

	LBWB (default)	WB	SB	SWB	AWB	MB
Serial correlation	✔		✔	✔	✔	✔
Heteroskedasticity	✔	✔		✔	✔	✔

ticity in error processes (see Table 1 for a summary). Among these methods, the LBWB accommodates nonstationary regressors and errors with a wide range of nonlinear dependence structures (Lin *et al.* 2024). It has been shown to perform consistently well across various practical scenarios and is therefore set as the default method in the package.

Local blockwise wild bootstrap

The local blockwise wild bootstrap (LBWB), recently proposed by Lin *et al.* (2024), accommodates the dynamics of regressors and error processes that are only locally stationary, encompassing a wide range of nonstationary processes with nonlinear dependence. It is based on the moving block bootstrap method proposed by Kunsch (1989) and captures the underlying dependence structure of $\{u_t, t \in \mathbb{Z}\}$ by resampling overlapping blocks of *oversmoothed* regression residuals local to each specific time point. These residuals are obtained using an oversmoothed bandwidth $\tilde{h} > h$; see Line 2 in Algorithm 1, Section 3. This step is crucial for nonparametric models as it helps to capture asymptotic bias, as demonstrated by Friedrich *et al.* (2020b) and Friedrich and Lin (2024). The choice of $\tilde{h}$ generally has minimal impact on the performance of the bootstrap methods, and thus **PyTimeVar** sets it following established practices, as recommended by Friedrich and Lin (2024), namely $\tilde{h} = Ch^{5/9}$ for some constant $C > 0$. The user can specify the value of C, otherwise $C = 2$ is set. As demonstrated in Lin *et al.* (2024), the LBWB provides accurate empirical coverage and performs more robustly than other common bootstrap methods across various simulation settings. The block length ℓ is automatically determined by **PyTimeVar** based on the sample size and bandwidth, following the rule used in Lin *et al.* (2024).

Wild bootstrap

The wild bootstrap (WB) is widely used in empirical studies similar to Model (1) due to its simplicity (see, e.g., Uddin, Mishra, and Smyth 2020; Awaworyi Churchill, Inekwe, Ivanovski, and Smyth 2020; Ren and Lucey 2023) and is known for its robustness against various forms of heteroskedasticity (Davidson and Flachaire 2008). It directly multiplies each of the oversmoothed regression residuals, as done in the LBWB, with a "wild" component that is an independent and identically distributed (i.i.d.) random variable with mean zero and variance one. Since these wild components are independent, serial correlation in error processes is disrupted in the bootstrapped world. Thus, the WB does not address serial correlation in errors. When there is strong evidence that errors are not serially correlated, the WB is recommended, as it generally produces shorter confidence intervals and bands compared to methods like the LBWB. Following standard practice, we use i.i.d. standard normal variables

to generate the wild components.

Sieve bootstrap

The sieve bootstrap (SB), as studied in Bühlmann (1998) and Friedrich and Lin (2024), is primarily used to address serial correlation in errors. The SB method involves fitting the oversmoothed residuals from the time-varying model with an AR(p) model and then resampling the *recentered* residuals from the estimated AR(p) model. The fundamental idea is that any purely deterministic and strictly stationary process with a zero mean can be approximated by an AR(∞) process, regardless of its linear dependence features (Kreiss, Paparoditis, and Politis 2011). While the SB method captures serial correlation, its applicability is limited to strictly stationary error processes. The SB method may be limited for more general classes of locally stationary processes because these processes are better approximated by AR processes with slowly increasing order and time-varying coefficients (Ding and Zhou 2023), rather than those with time-constant coefficients. In line with Bühlmann (1998, Section 5.1.1) and Friedrich and Lin (2024, Section 4.2), **PyTimeVar** uses the AIC as the default criterion to select the lag length p, but other common information criteria (BIC and HQIC) are available and can be specified by the user.

Sieve wild bootstrap

The sieve wild bootstrap (SWB) combines the wild bootstrap (WB) and sieve bootstrap (SB) methods, accommodating both serially correlated and heteroskedastic errors. This approach has been adopted for various applications, such as unit root testing (see, e.g., Cavaliere and Taylor 2009; Smeekes and Taylor 2012; Huang, Leng, Liu, and Peng 2020) and bias correction in dynamic panels (Chudik, Pesaran, and Smith 2023). In the SWB, each oversmoothed residual obtained from the AR(p) model is further multiplied by a "wild" component similar to the WB. Again, the wild components are generated using i.i.d. standard normal variables. As expected, the additional randomness results in the SWB yielding higher empirical coverage than the SB, albeit with a slightly larger empirical length, as shown in the simulation study by Lin *et al.* (2024).

Autoregressive wild bootstrap

The autoregressive wild bootstrap (AWB) was originally proposed by Smeekes and Urbain (2014) for multivariate unit root tests, with further discussion in Smeekes and Wilms (2023). Recently, Friedrich *et al.* (2020a) and Friedrich *et al.* (2020b) have applied the AWB to trend analysis of atmospheric ethane abundance. Further advancement in the AWB for nonparametric large panels is covered by Lin, van der Sluis, and Friedrich (2023). The AWB generates wild components using an AR(1) process and then multiplies these wild components directly with the oversmoothed regression residuals. This approach captures the serial correlation and heteroskedasticity of the error dynamics, mimicking the long-run variance of the limiting distribution of $\hat{\boldsymbol{\beta}}(\tau)$, where $\tau \in (0, 1)$. The AR(1) coefficient, denoted as $\gamma \in (0, 1)$, serves as a crucial tuning parameter in the AWB, analogous to the block length ℓ in the LBWB. As

discussed by Smeekes and Urbain (2014, p. 8) and Lin *et al.* (2023, Remark 1), the tuning parameter γ functions similarly to a kernel in estimating long-run variances and can be related to the block length ℓ in the LBWB. We adopt the conventional formula to express γ as a function of ℓ, as recommended by Smeekes and Urbain (2014) and Friedrich *et al.* (2020b).

Multiplier bootstrap based on long-run covariance matrix estimation

As discussed in Lin *et al.* (2024), the aforementioned bootstrap methods are residual-based, relying on careful resampling designs to accurately capture the dynamics of error processes. The nuisance parameters are automatically accounted for by the bootstrap without requiring additional estimation. In contrast, the multiplier bootstrap (MB) proposed by Zhou and Wu (2010) necessitates correcting asymptotic bias using a jackknife estimator, followed by obtaining consistent estimates of asymptotic long-run covariance matrices to correctly mimic the asymptotic distribution. This approach is similar to the simulation-based method proposed in Lin and Reuvers (2024). Similar ideas have also been considered in various settings, see, for instance, Wu and Zhou (2017), Li and Zhao (2019), Karmakar, Richter, and Wu (2022), and Gao, Peng, Wu, and Yan (2024). Note that the MB relies on a powerful Gaussian approximation result, accommodating a wide range of nonlinear, nonstationary processes with serial correlation and heteroskedasticity. However, the empirical coverage accuracy of the MB heavily depends on an accurate estimation of the long-run covariance matrices, which reflect the dependence structure. As reported by Friedrich and Lin (2024) and Lin *et al.* (2024), the MB can exhibit severe undercoverage when $\tau \in (0, 1)$ is near the boundaries or when n is small, due to the inherent challenges in estimating long-run covariance matrices in these scenarios. Despite these challenges, the theoretical robustness of this approach makes it a valuable option in **PyTimeVar** for practitioners seeking benchmark analysis.

Constructing bootstrap-based confidence intervals and bands

As mentioned, both pointwise confidence intervals and simultaneous confidence bands are crucial for making empirical conclusions. The MB method produces simultaneous bands that are therefore also valid pointwise. For the residual-based bootstrap methods, $(1 - \alpha)$-level pointwise confidence intervals can be easily constructed as shown in Algorithm 3 (Section 3), while obtaining simultaneous bands requires further consideration. Here, we follow the three-step procedure initially proposed by Bühlmann (1998) and commonly employed (see, e.g., Friedrich *et al.* 2020b; Friedrich and Lin 2024; Lin *et al.* 2023, 2024). The function `confidence_bands()` returns both intervals and bands and displays these automatically in a figure if `plots=True`.

2.4. Alternative estimation methods

For convenient comparison, we include several additional methods relevant to Model (1): the boosted Hodrick-Prescott (bHP) filter, power-law model, state-space model, and score-driven model. As mentioned in the Introduction, these methods may be available in various programming languages, and interested readers with more advanced analytical needs are

Table 2: Comparison of the different estimation methods

	LLE	bHP	power-law	state-space	score-driven
Trend analysis	✔	✔	✔	✔	✔
Time-varying relationship	✔			✔	✔

referred to the corresponding references listed there. Here, we focus on basic constructions to facilitate comparison and to unify the evaluation of different popular approaches. Note that the bHP filter and power-law model are designed solely for extracting trending patterns, meaning it applies only when $d = 0$ in Eq. (1). In contrast, the state-space and score-driven models support both trend analysis and regression with time-varying coefficients, similar to the local linear estimation discussed in Section 2.2. An overview of these methods is provided in Table 2.

Boosted HP filter for trend analysis

For $d = 0$, meaning $\boldsymbol{x}_t = 1$, the model simplifies to $y_t = \beta_{0,t} + u_t$, where $\beta_{0,t}$ represents the long-run trend and u_t captures short-run fluctuations, often interpreted as cyclical components in macroeconomic analysis. To estimate the long-run trend, an alternative nonparametric approach is the Hodrick-Prescott (HP) filter, which is widely used in macroeconomics for analyzing business cycles. The HP estimate of $\beta_{0,t}$ is obtained by solving the following minimization problem:

$$\hat{\beta}_{0,t}^{\mathrm{HP}} = \underset{\beta_{0,t}}{\operatorname{argmin}} \left\{ \sum_{t=1}^{n} (y_t - \beta_{0,t})^2 + \lambda \sum_{t=2}^{n} \left(\Delta^2 \beta_{0,t}\right)^2 \right\}, \qquad \lambda \geq 0, \tag{5}$$

where $\Delta^2 \beta_{0,t} = \Delta\beta_{0,t} - \Delta\beta_{0,t-1} = \beta_{0,t} - 2\beta_{0,t-1} + \beta_{0,t-2}$, and $\Delta\beta_{0,t} = \beta_{0,t} - \beta_{0,t-1}$ The second term in the equation penalizes the smoothness of the estimated trend. The parameter λ controls this smoothness: as $\lambda \to \infty$ the trend becomes linear, while as $\lambda \to 0$, the trend contains short-run fluctuations. Therefore, choosing an appropriate λ is crucial. While various optimization criteria are available for tuning parameters in other nonparametric methods, in macroeconomic literature, the tuning parameter is typically selected through empirical experimentation alone. Common practice sets, for instance, $\lambda = 1,600$ for quarterly data, irrespective of sample size. This choice may leave substantial trend dynamics in the residuals $\hat{u}_t^{\mathrm{HP}} = y_t - \hat{\beta}_{0,t}^{\mathrm{HP}}$. To overcome this issue, we recommend the method of boosted HP filter proposed by Phillips and Shi (2021), Biswas *et al.* (2024), and Mei *et al.* (2024), which can be called using the '`BoostedHP`' class in the **PyTimeVar** package. The bHP filter combines the HP filter with L_2-boosting. The boosting process involves iteratively refining the predictor (or learner) to remove leftover trend elements. Initially, we fit a simple learner to the data, compute the residuals, and then iteratively update the learner using these residuals until a stopping criterion is met. The articles mentioned above show that the resulting estimator successfully recovers various underlying trend dynamics as the number of iterations increases with the sample size. Two types of stopping criteria are considered in **PyTimeVar**. The first

involves an ADF unit root test to determine whether additional iterations are needed to adequately extract the trend component from the residuals. The second type employs information criteria to balance the tradeoff between the sample fit and the number of iterations.

Power-law trend model

The bHP filter discussed above is highly flexible for estimating trends. However, empirical analyses, such as trend analysis in climatic studies, often require structural interpretations of the trending behavior, such as the growth rate of greenhouse gases or temperature (Mudelsee 2019). In such cases, a parametric approach can be useful. For this reason, we incorporate a recently developed model introduced by Robinson (2012) and Lin and Reuvers (2024), referred to as the power-law trend model, for trend analysis in the **PyTimeVar** package. The basic idea is as follows: if the trend component $\beta_{0,t}$ is sufficiently smooth, it can be approximated using polynomial basis functions, where the polynomial powers are nonnegative integers. To increase flexibility, however, one could allow for *noninteger* powers, treat them as unknown, and let the data determine their values. Specifically, for $t = 1, \ldots, n$, the power-law model imposes a parametric structure on $\beta_{0,t}$ as

$$\beta_{0,t} = \sum_{i=1}^{p} \tau_i \, t^{\gamma_i}, \qquad p \in \mathbb{Z}^+, \tag{6}$$

where $\boldsymbol{\tau} := (\tau_1, \tau_2 \ldots, \tau_p)^\top$ and $\boldsymbol{\gamma} := (\gamma_1, \gamma_2, \ldots, \gamma_p)^\top$ are vectors of unknown parameters to be estimated. Note that γ_i for $i = 1, \ldots, p$ are not required to be integers. As mentioned, allowing for unknown trend powers provides more flexibility compared to conventional polynomial basis functions with integer powers. For identification, we assume that $\boldsymbol{\gamma} \in \boldsymbol{\Gamma} \subset \mathbb{R}^p$, where

$$\boldsymbol{\Gamma} = \Big\{ (\gamma_1, \gamma_2, \ldots, \gamma_p)^\top : \; -1/2 < \gamma_L \leq \gamma_1; \; \gamma_j - \gamma_{j-1} \geq \delta, \; j = 2, \ldots, p; \; \gamma_p \leq \gamma_U < \infty \Big\},$$

for some $[\gamma_L, \gamma_U] \subset (-1/2, \infty)$ and $\delta > 0$. Following Robinson (2012) and Lin and Reuvers (2024), we estimate $(\boldsymbol{\gamma}, \boldsymbol{\tau})$ though nonlinear least squares (NLS):

$$(\widehat{\boldsymbol{\gamma}}, \widehat{\boldsymbol{\tau}}) = \underset{(\boldsymbol{\theta}, \boldsymbol{\tau}) \in \boldsymbol{\Theta} \times \mathbb{R}^p}{\operatorname{argmin}} \; \frac{1}{2} \sum_{t=1}^{n} \left(y_t - \sum_{i=1}^{p} \tau_i \, t^{\gamma_i} \right)^2. \tag{7}$$

The function `PowerLaw()` implements the NLS estimation described above. In practice, the number of power-law trends, p, must be specified. Currently, no theoretical guidance exists for selecting this parameter. Users are advised to visually inspect the data's trending behavior and select the smallest p that sufficiently describes the data. The default is set to $p = 2$ in the package.

State-space model

State-space representations offer an alternative approach for extracting trend patterns and time-varying relationships (Durbin and Koopman 2012). The primary tools in this framework are the Kalman filter and smoother. Notably, there are some state-space specifications in

which the Kalman filter can reproduce the HP filter (see, e.g., Harvey and Trimbur 2008, and references therein). The state equation of the general univariate linear Gaussian state-space model can be expressed as:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{T}\boldsymbol{\beta}_t + \boldsymbol{R}\boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}), \tag{8}$$

where $\boldsymbol{\beta}_t$ is an unobserved vector, known as the state vector within the context of the state-space model and defined as in (1). The matrices $\boldsymbol{T}$ and $\boldsymbol{R}$ are assumed to be specified by the user, with the default setting being identity matrices. The observation equation remains identical to (1). Additionally, the error terms u_t and $\boldsymbol{\eta}_t$ are assumed to be Gaussian, serially uncorrelated, and independent of each other at all time points. The variances of u_t and $\boldsymbol{\eta}_t$, namely, σ_u^2 and $\boldsymbol{Q}$, respectively, can either be specified by the user or estimated through maximum likelihood estimation if not provided. The Kalman filter and smoother recursions presented in Algorithm 5 and 6 form the core methodological framework. These recursions facilitate the update of the system's state estimates as new observations are incorporated, thereby refining the knowledge of the system. We refer to Durbin and Koopman (2012) for more details. The '`Kalman`' class implements both the Kalman filter and smoother in the linear Gaussian state-space representation.

Score-driven model

Another popular approach to modeling time variations is score-driven models, often referred to as generalized autoregressive score (GAS) models, independently proposed by Creal *et al.* (2013) and Harvey (2013). The main difference between state-space and score-driven models is that in score-driven models, $\{\boldsymbol{\beta}_t\}$ is perfectly predictable one step ahead (see Eq. (9) below) given past information. In contrast, in state-space models, $\{\boldsymbol{\beta}_t\}$ cannot be predicted given past information, as $\boldsymbol{\eta}_t$ in Eq. (8) is independent of previous observations. For comprehensive summaries, see Harvey (2022) and Artemova, Blasques, van Brummelen, and Koopman (2022a,b). In a score-driven framework, parameters can be estimated using standard maximum likelihood (ML) techniques, regardless of whether u_t in Eq. (1) is Gaussian (see Blasques, van Brummelen, Koopman, and Lucas 2022 for theoretical analysis). However, for non-Gaussian u_t, the standard Kalman filter recursions fail. In such cases, other estimation techniques like simulated ML based on importance sampling or particle filtering can be employed, though these methods are more challenging and computationally intensive. An additional advantage of score-driven models over state-space models is their robustness to model mis-specifications. Specifically, even if the true dynamics of $\{\boldsymbol{\beta}_t\}$ follow a state-space form and the density of u_t is mis-specified, the path of $\{\boldsymbol{\beta}_t\}$ can still be consistently estimated. An extensive comparison between these two approaches can be found in Koopman, Lucas, and Scharth (2016), with theoretical justification provided by Beutner, Lin, and Lucas (2023).

The `fit()` function in our '`GAS`' class allows users to choose between $\mathcal{N}$-GAS, i.e., Gaussian GAS, and t-GAS models, depending on whether the density of u_t is specified as $\mathcal{N}(0, \sigma_u^2)$ or $t(0, \sigma_u^2, \nu)$, where $\sigma_u > 0$ is the scale parameter and $\nu > 0$ is the degree of freedom. The t-GAS model has the advantage of being robust to outliers and performing more stably, making it a popular choice (see, e.g., Harvey and Luati 2014; Gasperoni, Luati, Paci, and D'Innocenzo

2023; D'Innocenzo *et al.* 2023; Zou, Lin, and Lucas 2024). As $\nu \to \infty$, the construction of t-GAS reduces to $\mathcal{N}$-GAS.

Specifically, the score-driven approach assumes

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\omega} + \boldsymbol{B}\boldsymbol{\beta}_t + \boldsymbol{A}\boldsymbol{s}_t, \qquad \boldsymbol{s}_t = \boldsymbol{S}_t \boldsymbol{\nabla}_t, \qquad \boldsymbol{\nabla}_t = \frac{\partial \log p(y_t \mid \boldsymbol{x}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma})}{\partial \boldsymbol{\beta}_t}, \tag{9}$$

where $\boldsymbol{\omega}, \boldsymbol{B}, \boldsymbol{A}$ are static parameters to be estimated and are collected in the vector $\boldsymbol{\gamma} >$, $\boldsymbol{S}_t = \boldsymbol{S}(\boldsymbol{\beta}_t, \boldsymbol{\mathcal{D}}_t, \boldsymbol{\gamma})$ is a scaling matrix, and $\boldsymbol{\mathcal{D}}_t$ represents the data available up to time t, as defined in Section 2.1. Following common practice, let $\boldsymbol{B} = \text{diag}(B_0, \ldots, B_d)$ and $\boldsymbol{A} = \text{diag}(A_0, \ldots, A_d)$ be diagonal matrices, and collect all static parameters in $\boldsymbol{\gamma} = \left(\boldsymbol{\delta}^\top, \boldsymbol{\omega}^\top, B_0, \ldots, B_d, A_1, \ldots, A_d\right)^\top$, where $\boldsymbol{\delta} = \sigma_u$ for $\mathcal{N}$-GAS and $\boldsymbol{\delta} = (\nu, \sigma_u)^\top$ for t-GAS. Let $\boldsymbol{\Gamma}$ be a convex subset of $\mathbb{R}^{\dim(\boldsymbol{\gamma})}$. The vector of static parameters $\boldsymbol{\gamma}$ can be estimated through ML:

$$\hat{\boldsymbol{\gamma}} = \underset{\boldsymbol{\gamma} \in \boldsymbol{\Gamma}}{\text{argmax}}\, \mathcal{L}(\boldsymbol{\gamma}) = \underset{\boldsymbol{\gamma} \in \boldsymbol{\Gamma}}{\text{argmax}}\, \frac{1}{n} \sum_{t=1}^{n} \log p(y_t \mid \boldsymbol{x}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma}). \tag{10}$$

3. An Introduction to the PyTimeVar package

The proposed package, **PyTimeVar**, includes five classes that cater to different approaches as introduced in the previous section. In Section 3.1, we introduce the '`LocalLinear`' class, which implements local linear estimation and bootstrap-assisted inference. Section 3.2 presents the '`BoostedHP`' class, designed for the boosted HP filter method to extract deterministic trending patterns. Section 3.3 provides an overview of the '`PowerLaw`' class, which implements the power-law method for parametric trend analysis. Section 3.4 covers the '`Kalman`' class, which is built around Kalman filtering and smoothing based on state-space representations. In Section 3.5, we introduce the '`GAS`' class, which implements the score-driven approach. An overview of the package's functionalities is provided in Tables 3 and 4. Additional detailed algorithms for implementing the methods are provided in the appendix A.

We illustrate the package's functionalities with a running example. For this, we use the `'temperature'` dataset, which contains average yearly temperature changes across different world regions and can be accessed directly from the package. More details on the available datasets from our package will introduced in Section 3.6. Here, we utilize the global average data with the following code:

```
>>> from PyTimeVar.datasets import temperature
>>> import numpy as np
>>> data = temperature.load(regions=['World'],
...   start_date='1961', end_date='2023')
>>> vY = data.values
>>> mX = np.ones_like(vY)
>>> # set seed
>>> np.random.seed(123)
```

3.1. Local linear kernel estimation and bootstrap-assisted inference

We begin by introducing the functionalities for local linear estimation in the package, followed by an overview of bootstrap-assisted inference.

Implementation of local linear kernel estimation

The **PyTimeVar** provides the '`LocalLinear`' class to estimate time-varying trends and coefficients. The usage instructions are as follows:

```
>>> from PyTimeVar import LocalLinear
>>> model = LocalLinear(vY=vY, mX=mX)
>>> betaHatLLR = model.fit()

No bandwidth or selection method is specified.

============================================================
Optimal bandwidth selected by individual method:
- AIC method: 0.4286
- GCV method:  0.4286
- LMCV-0 method:  0.0600
- LMCV-2 method:  0.1150
- LMCV-4 method:  0.0600
- LMCV-6 method:  0.0600
============================================================
Optimal bandwidth used is the avg. of all methods:  0.1920
============================================================
Note: (1) For constructing confidence intervals/bands using the
residual-based bootstrap method, the avg. bandwidth of all
methods  0.1920 is used; (2) For constructing confidence intervals/bands
using the MB method, the GCV bandwidth  0.4286 is used.
```

Key parameters include the response variable, stacked in the vector `vY`, and the regressor matrix `mX` (or a vector of ones for trend analysis). Several parameters are optional to the '`LocalLinear`' class. First, the bandwidth parameter h can be set by the user via the `h` parameter; if not specified, an optimal bandwidth will be selected by averaging across all implemented bandwidth selection methods as suggested in Friedrich and Lin (2024). The user can specify the bandwidth selection method using the `bw_selection` parameter. Available options include LMCV-l (specified as `lmcv_l`, e.g., `bw_selection= 'lmcv_2'` for bandwidth selected by LMCV-2), as well as GCV (`bw_selection= 'gcv'`) and AIC (`bw_selection= 'aic'`). To average across all methods (denoted as AVG hereafter), namely LMCV-l for $l = 0, 2, 4, 6$, GCV, and AIC, users can specify `bw_selection='all'`. Note that for residual-based bootstrap methods, Friedrich and Lin (2024) and Lin *et al.* (2024) recommend using either the default or LMCV-6 methods. For the non-residual-based multiplier bootstrap

proposed by Zhou and Wu (2010), GCV is the recommended approach. Therefore, we suggest users follow these guidelines. We allow the users to specify the lower and upper bound for the bandwidth selection using parameters `LB_bw` and `UB_bw`, respectively. The default lower bound is set to `LB_bw = 0.06`. The default upper bound is set to `UB_bw=0.2` for LMCV, and to `UB_bw=0.7` for the AIC and GCV. Second, different kernel function options can be chosen, including Epanechnikov (default), Gaussian, quartic, triangular, tricube, and uniform.

The following example employs the local linear estimation with the LMCV-8 bandwidth selection method with the Gaussian kernel.

```
>>> model2LLR = LocalLinear(vY=vY, mX=mX, kernel='Gaussian',
...   bw_selection='lmcv_8')
>>> beta_hat_model2 = model2LLR.fit()

- LMCV-8 method:  0.0750
Optimal bandwidth used is lmcv_8:  0.0750
============================================================
Note: For constructing confidence intervals/bands using the MB method,
a GCV bandwidth is recommended.
```

We refer interested readers to Table 4 for an overview of all available function options. Moreover, estimation results can be obtained using `model.fit()`. The `model.summary()` function provides a table summarizing key results such as the selected data-driven bandwidth. In the running example, the summary is as follows.

```
>>> model.summary()

Local Linear Regression Results
============================================================
Kernel: epanechnikov
Bandwidth selection method: AVG
Bandwidth used for estimation:  0.1920
Number of observations: 63
Number of predictors: 1
============================================================
Beta coefficients (shape: (1, 63)):
Use the 'plot_betas()' method to plot the beta coefficients.
============================================================
Use the 'confidence_bands()' method to obtain the confidence bands and plots.
You can choose out of 6 types of Bootstrap to construct confidence bands:
SB, WB, SWB, MB, LBWB, AWB
============================================================
Use the 'plot_residuals()' method to plot the residuals.
```

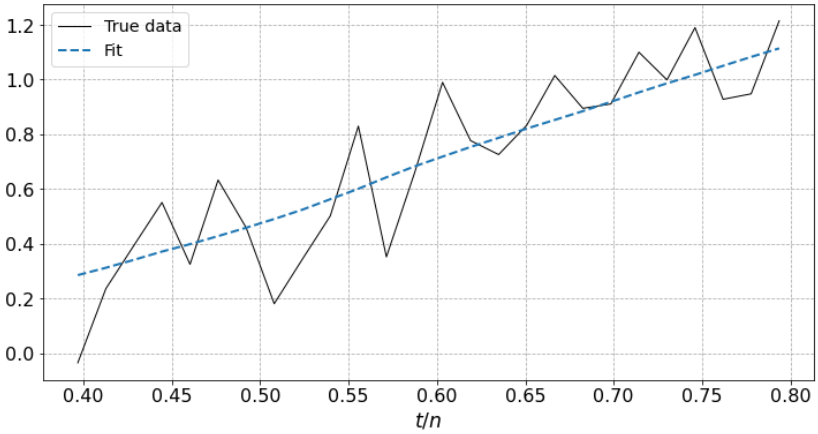


Figure 1: Estimated trend using *local linear estimation* (blue line) compared with the data (black line) for the temperature dataset. The x-axis represents the time ratio ranging from $[0.4, 0.8]$.

We offer visualizations of the estimates through the functions: (1) `plot_betas(tau)`, which plots the estimated coefficients $\boldsymbol{\beta}_t$ over time; (2) `plot_predicted(tau)`, which compares the true data with the fitted values; and (3) `plot_residuals(tau)`, which displays the residuals. The parameter `tau` allows users to "zoom in" on the graphs by specifying the ratio of the sample time period for visualization. It should be provided as a list containing two elements: the first element indicating the starting point and the second element defining the endpoint. If not specified, the figures show the results over the full interval $(0, 1)$. The example below shows the estimated trend compared to the actual data over the time interval $[0.4, 0.8]$ for the temperature example using the `plot_predicted()` function, with the outputs displayed in Figure 1.

```
>>> model.plot_predicted(tau=[0.4, 0.8])
```

Implementation of bootstrap-assisted inference

For inference, we offer six bootstrap methods as described in Section 2. These methods are divided into two categories: (1) for *residual-based* bootstrap, with detailed steps outlined in Algorithm 1, which also uses Algorithm 3 from Appendix A.1 to construct pointwise intervals and simultaneous bands; (2) and for the *non-residual-based* multiplier bootstrap, the procedure is described in Algorithm 2.

Several comments on Algorithm 1 are as follows. First, Line 2 uses a larger bandwidth $\tilde{h}$ for the bootstrap methods to capture asymptotic bias (Friedrich and Lin 2024). We follow the recommendation of Friedrich and Lin (2024), setting $\tilde{h} = Ch^{5/9}$, where h is the same bandwidth used in the local linear estimation described earlier. The constant C can be user-defined; otherwise, we set $C = 2$. Second, for the block length in the LBWB mentioned in Line 5, we use $\ell = \lfloor 4.5(nh)^{1/4} \rfloor$, as suggested by the theory. Third, for selecting the lag

Algorithm 1: Multiple residual-based bootstrap algorithms

for $t \in \{1, \ldots, n\}$ **do**
  Estimate Model (1) using a larger bandwidth $\tilde{h}$ than h, obtain the estimates $\tilde{\boldsymbol{\beta}}(\tau_t)$, and compute the residuals $\hat{u}_t = y_t - \tilde{\boldsymbol{\beta}}(\tau_t)^\top \boldsymbol{x}_t$;
end
if LBWB **then**
  Create a collection $\mathcal{U} = \{U_i,\, i = 1, \ldots, n - \ell + 1\}$, where each $U_i = (\hat{u}_i, \ldots, \hat{u}_{i+\ell-1})$ is a block of residuals with a block length $\ell < n$;
  for $b \in \{1, \ldots, B\}$ **do**
    for $i \in \{1, \ldots, \lceil n/\ell \rceil\}$ **do**
      Let $\mathcal{U}_i = \{U_{i\ell-2\ell+1}, \ldots, U_{i\ell+1}\} \cap \mathcal{U}$. Select randomly one block $U_{\pi(i)}$ from $\mathcal{U}_i$, where $\pi(i) \in \{i\ell - 2\ell + 1, i\ell - 2\ell + 2, \ldots, i\ell + 1\} \cap \{1, \ldots, n - \ell + 1\}$; Obtain $U_i^* = \xi_i^* U_{\pi(i)}$ with $\xi_i^* \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$;
    end
    Generate bootstrap innovations $u_1^*, \ldots, u_n^*$ by laying out $U_1^*, \ldots, U_N^*$;
    Compute $y_t^* = \tilde{\boldsymbol{\beta}}(\tau_t)^\top \boldsymbol{x}_t + u_t^*$, with $\{(y_t^*, \boldsymbol{x}_t)\}$, and obtain the bootstrap estimator $\hat{\boldsymbol{\beta}}^*(\cdot)$ using h;
  end
else if SB **then**
  Fit an AR(p) model to $\{\hat{u}_t\}$ with p selected by some criterion. Compute $\hat{\varepsilon}_t = \hat{u}_t - \sum_{i=1}^p \hat{\phi}_i \hat{u}_{t-i}$;
  for $b \in \{1, \ldots, B\}$ **do**
    Recenter $\hat{\varepsilon}_t$ to get $\tilde{\varepsilon}_t$, then resample $\{\tilde{\varepsilon}_t\}$ with replacement to form $\{\varepsilon_t^*\}$;
    Obtain $u_t^* = \sum_{i=1}^p \hat{\phi}_i u_{t-i}^* + \varepsilon_t^*$;
    Then follow Line 11;
  end
else if SWB **then**
  Follow the same as from Lines 14 - 19, whereas Line 16 is replaced by: Simulate $\varepsilon_t^* = \nu_t^* \hat{\varepsilon}_t$ with $\nu_t^* \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$;
else if WB **then**
  for $b \in \{1, \ldots, B\}$ **do**
    Simulate $u_t^* = \nu_t^* \hat{u}_t$ with $\nu_t^* \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$;
    Then follow Line 11;
  end
else if AWB **then**
  for $b \in \{1, \ldots, B\}$ **do**
    Simulate ξ_t^* with $\xi_t^* = \gamma \xi_{t-1}^* + \nu_t^*$ with $\nu_1^*, \ldots, \nu_n^* \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1 - \gamma^2)$ and $\xi_1^* \sim \mathcal{N}(0, 1)$, where γ is needed to be chosen. Obtain $u_t^* = \xi_t^* \hat{u}_t$;
    Then follow Line 11;
  end
end
for $j \in \{0, \ldots, d\}$ **do**
  Compute $\hat{q}_{j,\alpha}(\tau) = \inf\{u \in \mathbb{R} :\ \mathbb{P}^*(\hat{\beta}_j^*(\tau) - \tilde{\beta}_j(\tau) \le u) \ge \alpha\}$;
end
Implement Algorithm 3 to construct pointwise intervals and simultaneous bands.

Algorithm 2: Multiplier bootstrap algorithm by Zhou and Wu (2010)

Determine an appropriate bandwidth h for estimating $\hat{\boldsymbol{\beta}}(\cdot)$;
Let $h_{\mathrm{J}} = 2h$ and construct the jackknife estimator $\hat{\boldsymbol{\beta}}_{\mathrm{J}}(\cdot) = 2\hat{\boldsymbol{\beta}}_{h_{\mathrm{J}}/\sqrt{2}}(\cdot) - \hat{\boldsymbol{\beta}}_{h_{\mathrm{J}}}(\cdot)$;
for $b \in \{1, \ldots, B\}$ **do**
    Generate Gaussian vectors $\boldsymbol{\nu}_t \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \boldsymbol{I}_{d+1})$ and compute $\sup_{0\le\tau\le1} |\boldsymbol{\mu}_{h_{\mathrm{J}}}(\tau)|$, where $\boldsymbol{\mu}_{h_{\mathrm{J}}}(\tau) = (nh_{\mathrm{J}})^{-1} \sum_{t=1}^{n} \boldsymbol{\nu}_t K^*\Big(\frac{\tau_t - \tau}{h_{\mathrm{J}}}\Big)$, and $K^*(x) = 2\sqrt{2}K(\sqrt{2}\,x) - K(x)$;
end
Calculate the estimated $(1-\alpha)$-th quantile of $\sup_{0\le\tau\le1} |\boldsymbol{\mu}_{h_{\mathrm{J}}}(\tau)|$, denoted as $\hat{q}_{1-\alpha}$;
for $t \in \{1, \ldots, n\}$ **do**
    Obtain $\hat{\boldsymbol{M}}(\tau_t) = \boldsymbol{S}_{n,0}(\tau_t^*)$, where $\boldsymbol{S}_{n,0}(\tau) = (nh)^{-1} \sum_{t=1}^{n} \boldsymbol{x}_t \boldsymbol{x}_t^\top K\Big(\frac{\tau_t - \tau}{h}\Big)$ and $\tau_t^* = \max\{h, \min(\tau_t, 1-h)\}$;
    Let $\gamma_n = \lambda_n + (m+1)/n$, where $(m, \lambda_n) = (\lfloor n^{2/7} \rfloor, n^{-1/7})$;
    if $\tau_t \in [\gamma_n, 1-\gamma_n] \subset (0,1)$ **then**
        Compute $\hat{\boldsymbol{\Lambda}}(\tau_t) = \sum_{i=1}^{n} \omega(\tau_t, \tau_i)\Delta_i$, where $\omega(\tau_t, \tau_i) = K\Big(\frac{\tau_i - \tau_t}{\lambda_n}\Big) / \sum_{i=1}^{n} K\Big(\frac{\tau_i - \tau_t}{\lambda_n}\Big)$, $\boldsymbol{\Delta}_i = \boldsymbol{Q}_i \boldsymbol{Q}_i^\top / (2m+1)$, $\boldsymbol{Q}_i = \sum_{j=-m}^{m} \boldsymbol{x}_{i+j} \hat{u}_{i+j}$, and $\hat{u}_i = y_i - \hat{\boldsymbol{\beta}}(\tau_i)^\top \boldsymbol{x}_i$;
    else if $\tau_t \in [0, \gamma_n]$ **then**
        $\hat{\boldsymbol{\Lambda}}(\tau_t) = \hat{\boldsymbol{\Lambda}}(\gamma_n)$;
    else
        $\hat{\boldsymbol{\Lambda}}(\tau_t) = \hat{\boldsymbol{\Lambda}}(1-\gamma_n)$;
    end
    Construct the $(d+1)$-dimensional column vector $\hat{\boldsymbol{\sigma}}(\tau_t)$ by extracting the diagonal elements of the square matrix $\hat{\boldsymbol{\Sigma}}(\tau_t) = \Big(\hat{\boldsymbol{M}}^{-1}(\tau_t) \hat{\boldsymbol{\Lambda}}(\tau_t) \hat{\boldsymbol{M}}^{-1}(\tau_t)\Big)^{1/2}$;
end
Construct $(1-\alpha)$-th level simultaneous bands of $\boldsymbol{\beta}(\cdot)$ as $\hat{\boldsymbol{\beta}}_{\mathrm{J}}(\cdot) \pm \hat{\boldsymbol{\sigma}}(\cdot)\hat{q}_{1-\alpha}$.

length p in Line 14, the options include AIC (default), BIC, and HQIC. Fourth, in Line 29, the tuning parameter γ for the AWB can be specified by the user. Otherwise, γ is set according to the recommendations of Smeekes and Urbain (2014) and Smeekes and Wilms (2023), using $\gamma = 0.01^{1/\ell}$, where ℓ is the same block length used for the LBWB. Lastly, Line 34 computes the 100α-th percentile of the B centered bootstrap statistics $\hat{\beta}_j^*(\tau) - \tilde{\beta}_j(\tau)$, where $\mathbb{P}^*$ denoting the probability measure conditional on the original samples. The user is allowed to specify B and α using parameters `B` and `alpha` respectively, with defaults $B = 1,299$ and $\alpha = 0.05$.

We now discuss the multiplier bootstrap procedure as outlined in Algorithm 2. Following the recommendation by Zhou and Wu (2010), we use the generalized cross-validation (GCV) to select a data-driven bandwidth in Line 1. In Line 2, the *jackknife* bias-corrected estimator $\hat{\boldsymbol{\beta}}_{\mathrm{J}}(\cdot)$ is used to eliminate second-order bias. This is a crucial distinction from the residual-based bootstrap in Algorithm 1, where asymptotic bias is automatically accounted for. The terms $\hat{\boldsymbol{\beta}}_{h_{\mathrm{J}}/\sqrt{2}}(\cdot)$ and $\hat{\boldsymbol{\beta}}_{h_{\mathrm{J}}}(\cdot)$ in Line 2 refer to local linear estimators, as detailed in Section 2.2, with bandwidths $h_{\mathrm{J}}/\sqrt{2}$ and h_{J}, respectively. According to Zhou and Wu (2010), the

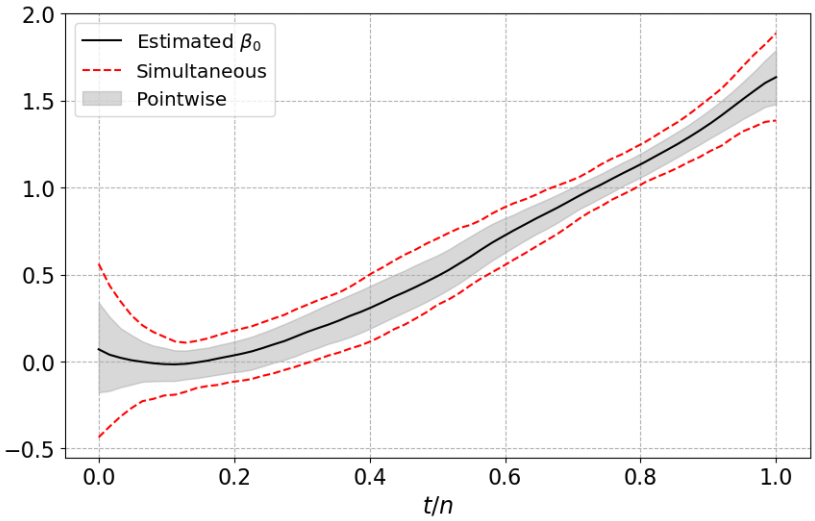


Figure 2: Estimated coefficient curve (black line) using local linear estimation, along with pointwise confidence intervals (gray area) and simultaneous confidence bands (red dotted line) for the temperature dataset. The x-axis represents the time ratio t/n, ranging from $[0, 1]$.

performance of the MB is relatively insensitive to the tuning parameters mentioned in Line 9, so we adhere to their rule-of-thumb provided in Section 4.4 of their work. Lines 7 through 18 construct an estimator for the long-run covariance matrix, as discussed on p. 8.

Bootstrap-assisted confidence pointwise intervals and simultaneous bands can be obtained using the function `model.confidence_bands()`. In Figure 2, we display the resulting LBWB confidence intervals and bands under the default settings:

```
>>> S_LB, S_UB, P_LB, P_UB = model.confidence_bands(bootstrap_type='LBWB',
...    Gsubs=None, plots=True)
```

All parameters in `confidence_bands()` are optional. Recall from Section 2.3 that both pointwise confidence intervals and simultaneous confidence bands can be constructed. When constructing confidence bands, the parameter `Gsubs` allows one to specify the subsets of $[0, 1]$ over which simultaneity is applied. For instance, to choose the interval $[0.4, 0.6]$, one can use `confidence_bands(Gsubs=[(int(0.4*n), int(0.6*n))])`. If `Gsubs` is not specified, the resulting simultaneous confidence bands will provide simultaneous coverage over the full sample $[0, 1]$. Note that the longer the subsets to be covered simultaneously, the wider the simultaneous bands typically become. Therefore, we recommend that users specify the intervals, as this can lead to more informative results and clearer interpretations. The six bootstrap methods available for `bootstrap_type` include LBWB (default), SB, SWB, WB, AWB and MB. If the boolean parameter `plots=True`, the `model.confidence_bands()` function generates plots of the pointwise confidence intervals and simultaneous confidence bands alongside the estimated coefficients. The output variables `S_LB` and `S_UB` store the lower and upper simultaneous bands, respectively, while `P_LB` and `P_UB` contain the lower and upper pointwise intervals, respectively, for further analysis.

3.2. Implementation of the boosted HP filter

In the 'BoostedHP' class, the function `fit()` is used to implement the boosted HP filter for trend extraction. The details of the bHP method are provided in Algorithm 4 (Appendix A.2). The function can be applied as follows:

```
>>> from PyTimeVar import BoostedHP
>>> bHPmodel = BoostedHP(vY=vY, dLambda=1600, iMaxIter=100)
>>> bHPtrend, bHPresiduals = bHPmodel.fit(
...   boost=True, stop='adf', dAlpha=0.05, verbose=False)
```

The minimum required input is `bHPmodel.fit(vY)`, where `vY` is the vector of the dependent variable. The smoothing parameter `dLambda` can be manually specified; otherwise, the default value $\lambda = 1600$ will be used as in Phillips and Shi (2021). Additionally, the maximum number of iterations for the boosting algorithm is set to 100 by default. The function `bHPmodel.fit()` includes the parameter `boost`, a Boolean that indicates whether to boost the original HP filter. If set to `False`, the standard HP filter is used. It returns the estimated trends for each boosting iteration along with the corresponding residuals. Additionally, the stopping criteria for the algorithm, as discussed in Section 2.4.1, can be specified using the `stop` parameter. The available options are `'adf'` (default), `'bic'`, `'aic'`, and `'hq'`, with the ADF unit root test being the default method. The parameter `dAlpha` sets the significance level for the stopping criterion, which is applicable only when using the ADF unit root test. The parameter `verbose` is a Boolean that controls whether a progress bar is displayed during execution. Additional details about the model can be accessed through `bHPmodel.summary()`, which includes the selected parameters, the information criteria values at each iteration, and the estimated trend from the bHP filter.

```
>>> bHPmodel.summary()

Boosted HP Filter Results
============================================================
Stopping Criterion: adf
Max Iterations: 100
Iterations Run: 1
============================================================
Lambda: 1600
Alpha: 0.05
Information Criteria Values: [3.54412124e-12]
============================================================
```

The function `bHPmodel.plot(tau)` plots the original series and the trend component. The parameter `tau` (default: $(0, 1)$) follows the same approach as demonstrated in Section 3.1.1. An example output is given below in Figure 3.

```
>>> bHPmodel.plot()
```

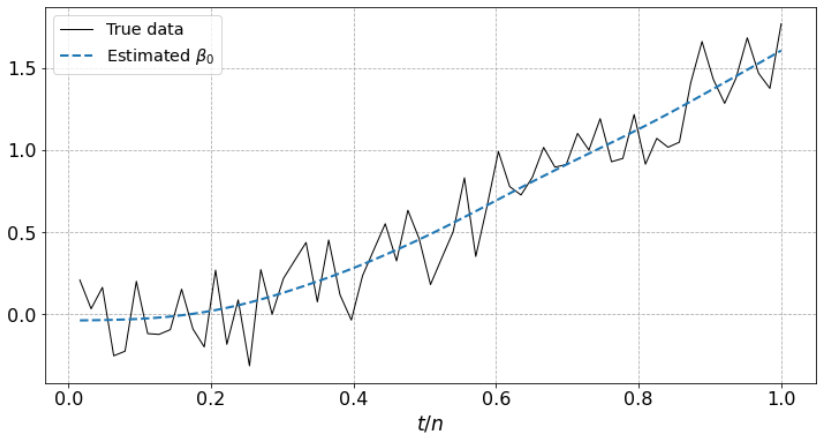


Figure 3: Estimated trend using the *boosted HP* (blue line) compared with the data (black line) for the temperature dataset. The x-axis represents the time ratio t/n, ranging from $[0, 1]$.

3.3. Implementation of the power-law model

The '`PowerLaw`' class implements the power-law model and includes three functions: `fit()` for calculating the power-law trend, `summary()` for displaying the fitted model's equation, and `plot()` for visualizing the fitted power-law trend. The minimum input for the '`PowerLaw`' instant is the data vector `vY`. The parameter `n_powers` can be specified to set the number of powers used to determine the trend, with the default value set to `n_powers=2` if not provided. The `fit()` function returns the estimated trend and parameter vector. An illustration of the '`PowerLaw`' class with a single power for the temperature dataset is shown in Figure 4.

```
>>> from PyTimeVar import PowerLaw
>>> PwrLaw = PowerLaw(vY=vY, n_powers=1)
>>> pwrTrend, pwrGamma = PwrLaw.fit()
>>> PwrLaw.summary()
>>> PwrLaw.plot()

Power-Law Trend Results:
==============================
yhat= 0.001 t^1.848
```

As shown in Eq. (7), determining the optimal parameters for the power-law model is a nonlinear optimization problem. In the package, we address this by first concentrating out $\boldsymbol{\theta}$ to obtain an optimizer $\boldsymbol{\tau}(\boldsymbol{\theta})$. Then, we compute the optimal values of $\boldsymbol{\theta}$ by minimizing the sum of squared residuals. We start the optimization process by fixing $\boldsymbol{\theta}$ as a vector $(0, 1, \ldots,$ `n_powers` $- 1)$. The user is allowed to set the initial values using the parameter `vgamma0`. Since we make use of the `scipy.optimize.minimize()` function, we allow the user to specify the same optimization options as in `scipy.optimize.minimize()`. These options

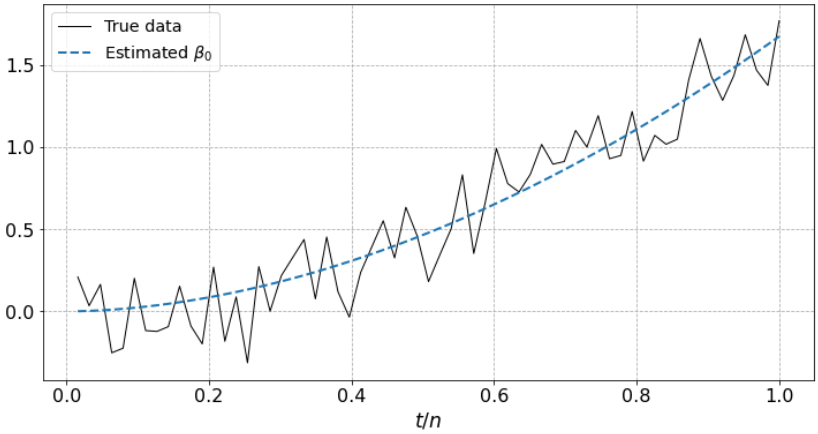


Figure 4: Estimated trend using the *power-law* model (blue line) compared with the data (black line) for the temperature dataset. The x-axis represents the time ratio t/n, ranging from $[0, 1]$.

include `maxiter` to specify the maximum number of iterations and the boolean parameter `disp` to indicate whether convergence messages are required. Furthermore, we provide users the option to specify bounds for parameter optimization. For example, the code below shows how to use these options when the user sets `n_powers=2`, specifies the first power as zero, searches for the second power law within the interval $[0.1, 5]$, limits the iterations to a maximum of 1,000, and disables the display of convergence messages.

```
>>> vgamma0 = np.arange(0, 0.1, 0.05)
>>> options = {'maxiter': 1E3, 'disp': False}
>>> bounds = ((0,0),(0.1, 5), )
>>> auxPwr = PowerLaw(vY, n_powers=2, vgamma0=vgamma0,
...   bounds=bounds, options=options)
```

3.4. Implementation of the state-space model

The '`Kalman`' class implements the Kalman filter and smoother based on Algorithms 5 and 6. Algorithm 5 assumes initially that $\boldsymbol{\beta}_1 \sim \mathcal{N}(\boldsymbol{b}_1, \boldsymbol{P}_1)$, where $\boldsymbol{b}_1$ and variance $\boldsymbol{P}_1$ are known. To initialize the model users can specify the scalar σ_u, vector $\boldsymbol{b}_1$, and matrices $\boldsymbol{T}$, $\boldsymbol{R}$, $\boldsymbol{Q}$, and $\boldsymbol{P}_1$, as well as an optional matrix of regressors `mX`. If a regressor matrix is not provided, the regressors will be set to a scalar $\boldsymbol{x}_t = 1$. If $\boldsymbol{Q}$ and/or σ_u are not specified, they will be estimated using maximum likelihood estimation. By default, $\boldsymbol{T}$ is set as an identity matrix, $\boldsymbol{R}$ and $\boldsymbol{P}_1$ are set to the scalar 1, and $\boldsymbol{b}_1$ is initialized as a zero vector. Users can choose to compute filtered state estimates, one-step-ahead predictions, or smoothed estimates using the function `Kalman.fit(option)`, where `option` can be `'filter'` (default), `'predictor'`, `'smoother'`, or `'all'`. The case of `Kalman.fit('all')` automatically provides the filter, predictor, and smoother together. An example with the temperature dataset is provided below.

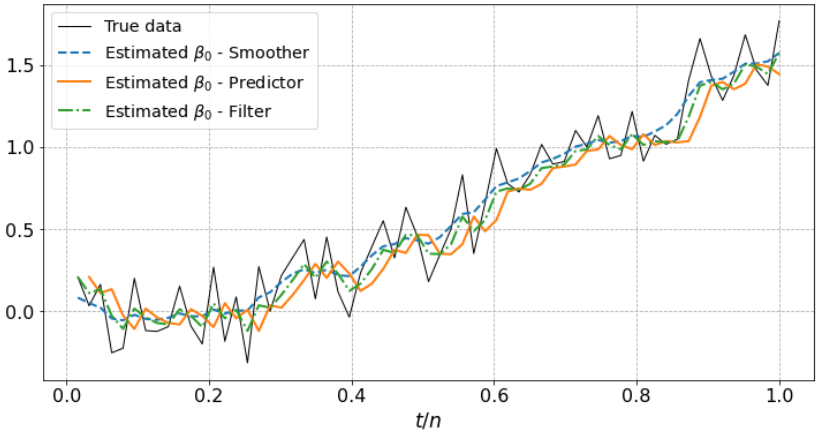


Figure 5: Estimated trend using the *Kalman smoother* (blue dashed), *predictor* (orange solid), and *filter* (green dash-dot), compared with the data (black solid) for the temperature dataset. The x-axis represents the time ratio t/n, ranging from $[0, 1]$.

```
>>> from PyTimeVar import Kalman
>>> kalmanmodel = Kalman(vY=vY)
>>> [kl_filter, kl_predictor, kl_smoother] = kalmanmodel.fit('all')
```

We provide two types of outputs. First, users can retrieve the specification of the linear Gaussian state-space model using the `summary()` function. Second, the `plot()` function enables users to visually display the estimated filter, predictions, and/or smoother alongside the data points. When all methods are applied (i.e., when `Kalman.fit('all')` is called), users can choose whether to plot the fitted estimates – filter, predictor, and smoother – in a single figure or separately, using the boolean parameter `individual`. By default, we set `individual=False` and thus the estimates are plotted in a single figure. The following code provides an example using the temperature dataset. The resulting plot is shown in Figure 5.

```
>>> kalmanmodel.plot(individual=False)
```

3.5. Implementation of the score-driven model

Users can choose between $\mathcal{N}$-GAS and t-GAS models. The implementation of GAS estimation is detailed in Algorithm 7 in Appendix A.4. As indicated in Eq. (9), the spectral norm of $\boldsymbol{B} = \text{diag}(B_0, \ldots, B_d)$ should be strictly less than 1 to ensure a strictly stationary and ergodic solution. Consequently, the bound constraints for B_j are suggested to be set within $(-1, 1)$ for $j = 0, \ldots, d$. In our package, the bounds for the static parameter vector $\boldsymbol{\gamma}$ are set automatically, unless specified by the user in the optional parameter `bounds`. The package allows users to specify bound constraints for all parameters. More technical details are available in Appendix A.4.

The '`GAS`' class implements the score-driven models described above. To obtain the estimated trend for temperature data and the estimated parameters for the $\mathcal{N}$-GAS model, for instance,

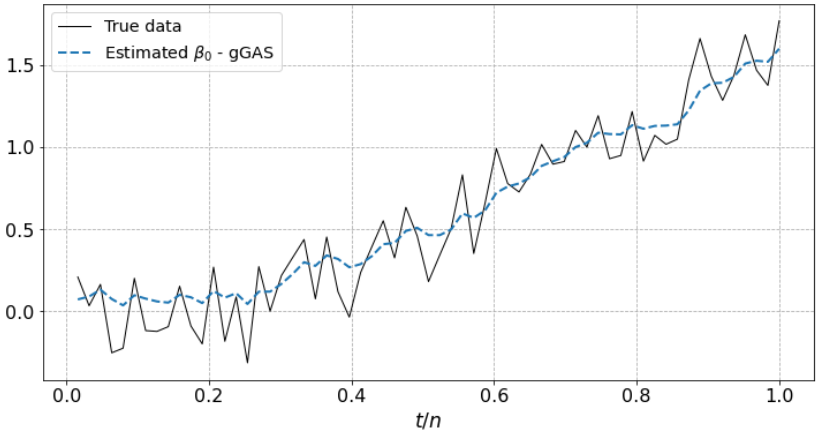


Figure 6: Estimated trend using $\mathcal{N}$-GAS models (blue line) compared with the data (black line) for the temperature dataset. The x-axis represents the time ratio t/n, ranging from $[0, 1]$.

we use the `GAS.fit()` function as follows:

```
>>> from PyTimeVar import GAS
>>> N_gasmodel = GAS(vY=vY, mX=mX, method='gaussian', niter=10)
>>> N_GAStrend, N_GASparams = N_gasmodel.fit()
```

Similar to `LocalLinear()`, the vector `vY` denotes the response variable, while `mX` represents the regressor matrix (or a vector of ones for trend analysis). The argument `vgamma0` is optional. If specified, it provides the initial estimate for the parameter vector $\boldsymbol{\gamma}_0$. The structure and dimensionality of $\boldsymbol{\gamma}_0$ depend on the chosen method. For the $\mathcal{N}$-GAS model, $\boldsymbol{\gamma}_0$ should be set as $\boldsymbol{\gamma}_0 = (\sigma_{u,0}, \boldsymbol{\omega}_0^\top, B_0, \ldots, B_d, A_0, \ldots, A_d)^\top$. For the t-GAS model, $\boldsymbol{\gamma}_0$ should include an initial value for the degree of freedom ν, i.e., $\boldsymbol{\gamma}_0 = (\nu_0, \sigma_{u,0}, \boldsymbol{\omega}_0^\top, B_0, \ldots, B_d, A_0, \ldots, A_d)^\top$. If `vgamma0` is not provided, the function `fit()` will compute an initial vector for $\boldsymbol{\gamma}_0$ automatically. The optional parameter `options` specifies the optimization settings, following the same usage as in the '`PowerLaw`' class. To avoid the potential risk of convergence to local optima, we use the basin-hopping optimization algorithm proposed by Wales and Doye (1997), implemented in Python by **scipy**. Users can achieve this by setting `niter` to be a positive integer.

Since the t-GAS model is known for its robustness to outliers, it is used as the default option if no specific method is provided. The estimated matrix of time-varying coefficients is saved in `GAS.betas`, while the estimated values of $\boldsymbol{\gamma}$ are stored in `GAS.params`. Moreover, each estimated time-varying coefficient will be plotted with the call of the function `GAS.plot()` as follows, and the resulting plot is demonstrated in Figure 6.

```
>>> N_gasmodel.plot()
```

3.6. Empirical datasets

In addition to the methods described above, our package offers a diverse set of datasets for practical exercises in trend analysis and time-varying regressions. These datasets provide rich temporal insights across various domains:

(i) Monthly U.S. inflation rate data from 01/02/1947 to 01/08/2024 ($n = 931$), calculated as the log-difference of seasonally adjusted Consumer Price Index, sourced from https://fred.stlouisfed.org/series/CPIAUCSL.

(ii) Daily USD index data from 20/01/1986 to 06/09/2024 ($n = 9,879$), sourced from https://www.wsj.com/market-data/quotes/index/DXY/historical-prices.

(iii) Yearly global average temperature data from 1961 to 2023 ($n = 63$), along with additional temperature data from specific regions, sourced from https://www.fao.org/faostat/en/#home.

(iv) Yearly CO_2 emission data for 19 early industrialized countries from 1900 to 2017 ($n = 118$), sourced from https://data.ess-dive.lbl.gov/portals/CDIAC.

(v) Financial herding data from China (sourced from Ren and Lucey 2023).

Further information on the available datasets can be found in the online documentation at https://pytimevar.readthedocs.io/en/latest/PyTimeVar.datasets.html.

4. Illustrative examples

In this section, we illustrate the functionalities of our package using the financial herding data from China as investigated in Ren and Lucey (2023) and Lin *et al.* (2024). Specifically, we adopt the following model to study time-varying herding effects across different time periods (Ren and Lucey 2023):

$$\begin{aligned}\text{CSAD}_{m,t} = \gamma_0(t/n) + \gamma_1(t/n)R_{m,t} \\ + \gamma_2(t/n)|R_{m,t}| + \gamma_3(t/n)R_{m,t}^2 + \gamma_4(t/n)\text{CSAD}_{m,t-1} + \varepsilon_t. \quad (11)\end{aligned}$$

Here, $\text{CSAD}_{m,t} = N^{-1}\sum_{i=1}^{N}|R_{i,t} - R_{m,t}|$ represents the cross-sectional absolute deviation of returns (CSAD), where $R_{i,t}$ is the return of stock i at time t, and $R_{m,t}$ denotes the average market return across N stocks at time t.

4.1. Trending patterns

We begin by examining the trend patterns in the measure $\text{CSAD}_{m,t}$ across different approaches. To visualize the estimated trends using local linear estimation, we utilize the `LocalLinear()` and `fit()` functions from the '`LocalLinear`' class. For trend extraction, `mX` is set as a column vector of ones, and the bandwidth is selected using the LMCV-6 with the Epanechnikov kernel, resulting in $h = 0.14$. Given the selected bandwidth, the code for performing the local linear estimation is shown below:

Table 3: Overview of the functions provided in `PyTimeVar`

Class	Function	Description
'LocalLinear'	`fit()`	fit the model by local linear estimation and return estimated coefficients `betahat`
	`summary()`	print a summary of local linear regression results, including bandwidth, number of observations, and beta coefficients
	`plot_betas()`	plot estimated coefficients curve over a normalized x-axis specified by the parameter `tau`. If `tau` is not provided, the curve is plotted over a normalized x-axis ranging from 0 to 1. The `tau` parameter follows the same usage across all `plot()` functions
	`plot_predicted()`	plot true data against fitted values over a time interval specified by the parameter `tau` (default: $(0, 1)$)
	`plot_residuals()`	plot residuals over a time interval specified by the parameter `tau` (default: $(0, 1)$)
	`confidence_bands()`	construct and plot bootstrap confidence intervals/bands for β_t. The following parameters can be specified: • `bootstrap_type`: bootstrap method (string), including `'LBWB'` (default), `'SB'`, `'SWB'`, `'MB'`, `'AWB'`, and `'WB'`. • `alpha`: significance level (float, optional) for constructing the confidence intervals/bands (default: 0.05) • `gamma`: tuning parameter only used in `'AWB'`(float, optional) • `ic`: information criterion for a data-driven selection of lag length in `'SB'` and `'SWB'` (string, optional, default: `'AIC'`) • `Gsubs`: a list of tuples indicating subsample ranges for constructing confidence intervals/bands (optional, default: full sample) • `Chtilde`: tuning parameter for the oversmoothing parameter $\tilde{h}$ used in all residual-based bootstrap (float, default: 2) • `B`: number of bootstrap iterations (integer). The default for the residual-bootstrap methods is 1,299, and 3000 for the MB. • `plots`: whether or not to plot results (boolean flag, default: False).
'BoostedHP'	`fit()`	fit by bHP filter and return trend estimates and residuals. The following parameters can be specified: • `boost`: whether or not to use the boosting (boolean flag, default: True) • `stop`: stopping criterion, including `'adf'` (default), `'bic'`, `'aic'`, and `'hq'` • `dAlpha`: significance level for the stopping criterion `'adf'` (float, default: 0.05) • `verbose`: whether or not to display progress (boolean flag, default: False)
	`summary()`	print a summary of bHP filter results, including stopping criterion, iterations run, λ, and α
	`plot()`	plot true data against estimated trend over a time interval specified by the parameter `tau` (default: $(0, 1)$)
'PowerLaw'	`fit()`	fit by power-law model and return trend estimates and parameter estimates
	`summary()`	print prediction equation
	`plot()`	plot true data against estimated trend over a time interval specified by the parameter `tau` (default: $(0, 1)$)
'Kalman'	`fit()`	compute the Kalman filtered states, one-step ahead predicted states or smoothed states, according to the specified `option`, including `'filter'` (default), `'predictor'`, `'smoother'`, and `'all'`
	`summary()`	print a summary of the state-space model specifications, including the values of matrices `H`, `Q`, `R` and `T`
	`plot()`	plot estimates against true data. The following parameters can be specified: • `tau`: the time interval for plots (default: $(0, 1)$) • `individual`: whether to plot the estimated results from the Kalman filter, predictor, and smoother in a single plot when `Kalman.fit('all')` is called (boolean flag, default: False)
'GAS'	`fit()`	fit score-driven model, according to the specified `method` (`'gaussian'` or `'student'`)
	`plot()`	plot estimated coefficients against true data over a time interval specified by the parameter `tau` (default: $(0, 1)$)

Table 4: Overview of the class constructors in the '`LocalLinear`', '`BoostedHP`', '`PowerLaw`', '`Kalman`', and '`GAS`' classes

Class/Constructor	Arguments
'`LocalLinear`'/`LocalLinear()`	• `vY`: dependent variable vector • `mX`: regressor matrix • `h`: bandwidth for regression (float, optional) • `kernel`: kernel (string) used in estimation; possible choices include `epanechnikov`(default), `'gaussian'`, `'uniform'` `'triangular`, `'quartic'`, and `'tricube'` • `bw_selection`: bandwidth selection method (string, optional); possible choices include `'all'` (default), `'aic'`, `'gcv'`, `'lmcv_l'` • `LB_bw`: lower bound for data-driven bandwidth (float, optional, default: 0.06) • `UB_bw`: upper bound for data-driven bandwidth (float, optional, default: 0.2 for `lmcv_l`; 0.7 for `aic` and `gcv`)
'`BoostedHP`'/`BoostedHP()`	• `vY`: dependent variable vector • `dLambda`: smoothing parameter (float, default: 1600) • `iMaxIter`: maximum number of iterations for boosting (integer, default: 100)
'`PowerLaw`'/`PowerLaw()`	• `vY`: dependent variable vector • `n_powers`: the number of powers (integer, default: 2) • `vgamma0`: initial value vector (default: `[0, 1, ..., n_powers-1]`) • `options`: stopping criteria (dict, default: `{'maxiter': 5E5}`)
'`Kalman`'/`Kalman()`	• `vY`: dependent variable vector • `mX`: regressor matrix (optional) • `R`: transition correlation matrix (optional) • `T`: transition matrix (optional) • `Q`: transition covariance matrix (optional) • `sigma_u`: observation noise variance (optional) • `b_1`: initial state mean vector (optional). • `P_1`: initial state covariance matrix (optional).
'`GAS`/'`GAS()`	• `vY`: dependent variable vector • `mX`: regressor matrix • `method`: method (string) to estimate GAS model; possible choice include `'student'` and `'gaussian'` (default: `'student'`) • `vgamma0`: initial parameter vector (optional) • `bounds`: parameter space (list, optional) • `options`: stopping criteria for optimization (dict, optional, default: `{'maxfun': 5E3}`) • `niter`: the number of basin-hopping iterations for the optimization function `scipy.optimize.basinhopping()` (integer, default: 10)

```
>>> from PyTimeVar import LocalLinear
>>> from PyTimeVar.datasets import herding
>>> herd_data = herding.load(start_date='2015-01-05',
...   end_date='2022-04-29')
>>> vY = herd_data[['CSAD_AVG']].values
>>> mX = np.ones_like(vY)
>>> LLr_model = LocalLinear(vY=vY, mX=mX, h=0.14)
>>> LLr_trend = LLr_model.fit()
```

Additionally, the functions `BoostedHP()`, `Kalman()`, `GAS()`, along with their respective `fit()` functions from the '`BoostedHP`', '`Kalman`', and '`GAS`'classes, are used to estimate the trend with the Boosted HP filter (with the ADF stopping criterion), Kalman smoother, and t-GAS model. The code for each of these methods is provided below.

```
>>> from PyTimeVar import BoostedHP
>>> bHPmodel = BoostedHP(vY=vY, dLambda=1600, iMaxIter=100)
>>> bHPtrend, bHPresiduals = bHPmodel.fit(boost=True, stop='bic',
...   dAlpha=0.05, verbose=False)

>>> from PyTimeVar import Kalman
>>> kalmanmodel = Kalman(vY=vY, mX=mX)
>>> Kalmansmooth_trend = kalmanmodel.fit('smoother')

>>> from PyTimeVar import GAS
>>> gasmodel = GAS(vY=vY, mX=mX, method='student')
>>> tGAStrend, tGASparams = gasmodel.fit()
```

We compare the trend estimates from the local linear estimation with those from the three alternative methods mentioned above. Figure 7 displays the data $\text{CSAD}_{m,t}$ along with the estimated trends from these four approaches. All methods effectively capture the trend pattern, with the local linear estimator providing the smoothest results as expected.

4.2. Time-varying herding effects

Next, we focus on illustrating our main tool, the local linear estimation combined with bootstrap-assisted inference. To demonstrate its usage, we replicate the empirical study of Lin *et al.* (2024), originally studied by Ren and Lucey (2023), applying the nonparametric approach to Eq. (11). Our primary focus is on the coefficient curve of γ_3 from Eq. (11), which serves as an indicator of local herding behavior. In addition to the full sample simultaneous confidence bands, we examine two subsets of the full sample to explain the usage when `Gsubs` are required: G1, the period preceding the first break date, and G6, the period following the fifth break date. Similar to identifying trending patterns, the '`LocalLinear`' class provides the functions `fit()` and `confidence_bands()` to estimate time-varying herding effects and construct confidence bands using local linear estimation and bootstrap methods. The LBWB

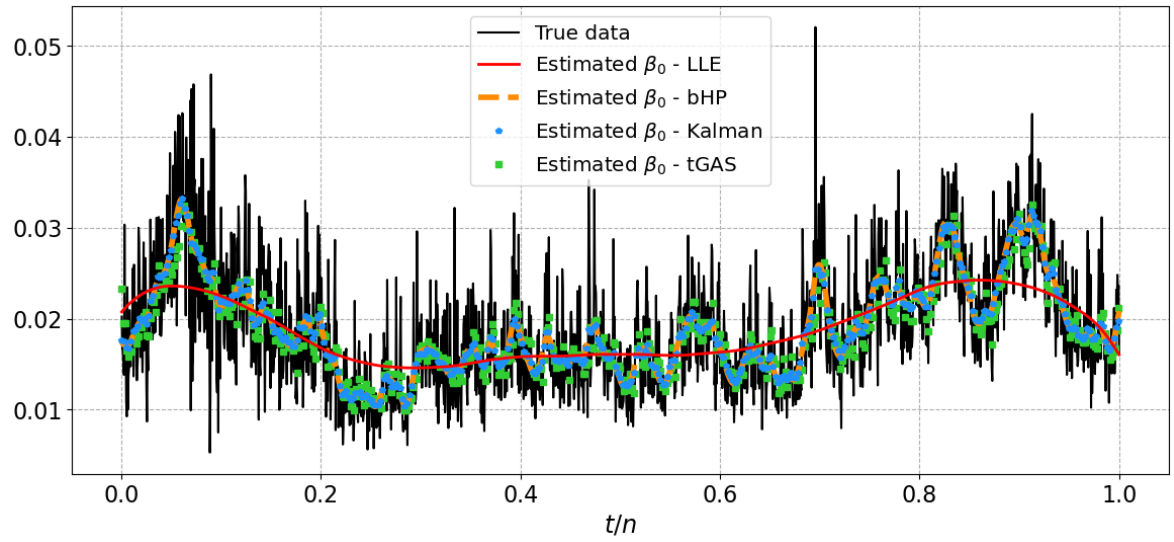
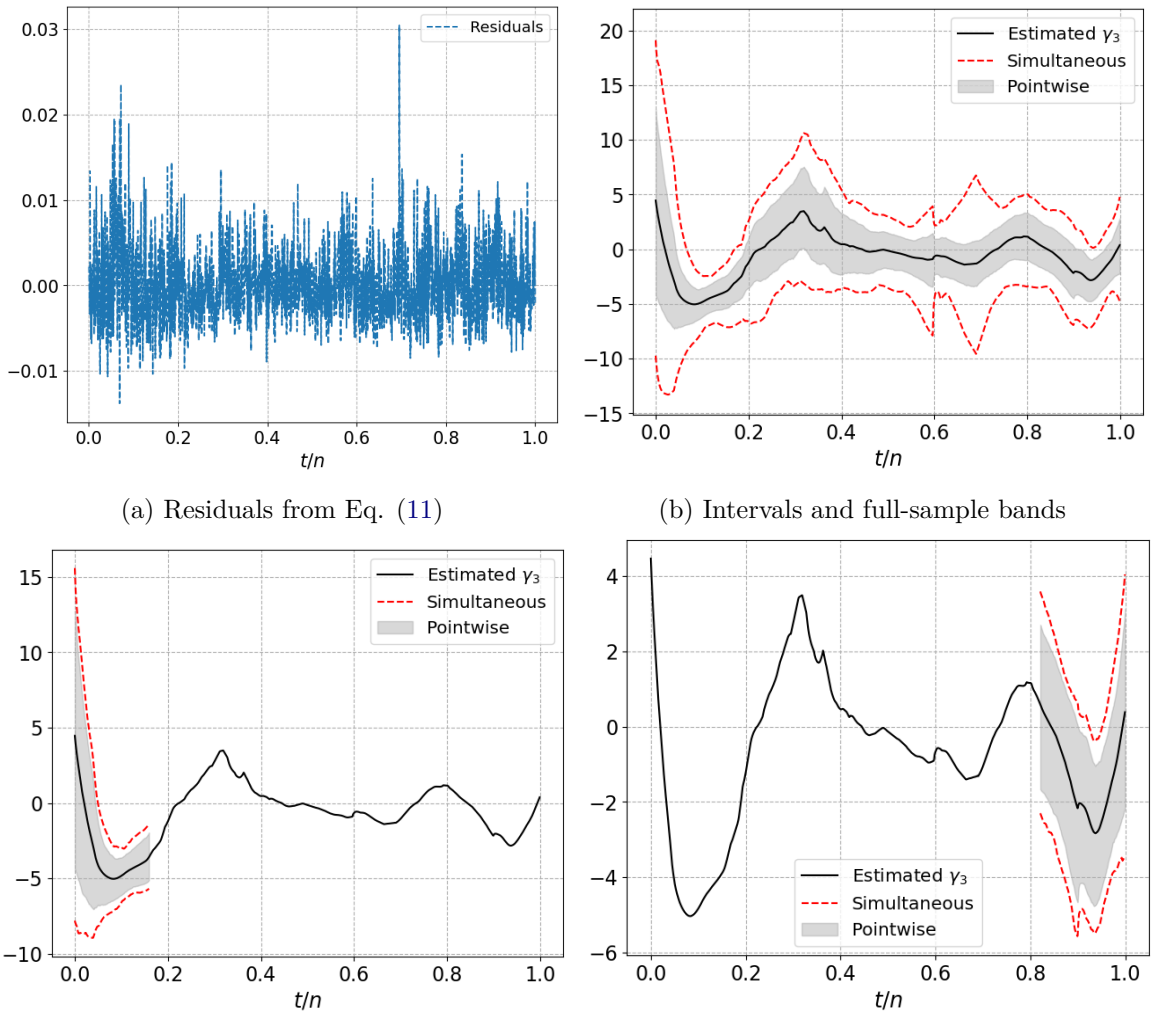


Figure 7: Estimated trends using local linear estimation (red solid), boosted HP filter (orange dashed), Kalman smoother (blue pentagonal), and t-GAS models (green square) compared with the data (black solid) for the herding dataset. The x-axis represents the time ratio t/n, ranging from $[0, 1]$.

method is selected for bootstrap due to its superior performance as found in Lin *et al.* (2024). We perform $1,299$ bootstrap replications. Additionally, the bandwidth is set to 0.0975, and the Epanechnikov kernel is used, following the recommended choices from Lin *et al.* (2024). The following code implements the procedures described above.

```
>>> from PyTimeVar.datasets import herding
>>> PyTimeVar import LocalLinear
>>> vY, mX = herding.load(data_replication=True)
>>> breakindex_1 = 284
>>> breakindex_5 = 1459
>>> LLr_model = LocalLinear(vY=vY, mX=mX, h=0.0975)
>>> LLr_betahat = LLr_model.fit()
>>> residuals = LLr_model.plot_residuals()
>>> S_LB_full, S_UB_full, P_LB_full, P_UB_full = LLr_model.confidence_bands(
...   plots=True)
>>> S_LB_G1, S_UB_G1, P_LB_G1, P_UB_G1 = LLr_model.confidence_bands(
...   plots=True,
...   Gsubs=[(0, breakindex_1)])
>>> S_LB_G6, S_UB_G6, P_LB_G6, P_UB_G6 = LLr_model.confidence_bands(
...   plots=True,
...   Gsubs=[(breakindex_5, len(vY))])
```

We set the argument `plots=True` in the `confidence_bands` function to display the LBWB



(a) Residuals from Eq. (11)

(b) Intervals and full-sample bands

(c) Intervals and bands based on subsample G1

(d) Intervals and bands based on subsample G6

Figure 8: The estimated $\gamma_3(\cdot)$ using local linear estimation, along with the 95%-level confidence intervals and bands, obtained using the LBWB bootstrap.

confidence band plots and specify the required subsets in `Gsubs`. The lower and upper bounds of the bootstrap-assisted simultaneous bands are saved in `S_LB` and `S_UB`, respectively, while the pointwise intervals are stored in `P_LB` and `P_UB`, respectively. The replication results are presented in Figure 8, which shows the simultaneous confidence bands for $\hat{\gamma}_3(\cdot)$ for the full sample, G1, and G6. These are detailed in Figure 8b, Figure 8c, and Figure 8d, respectively. Additionally, the residuals from Eq. (11) are displayed in Figure 8a. In Appendix A.5, we apply additional bootstrap methods as in Figure 8b. The observed patterns show no qualitative differences in this example.

5. Conclusion

Time-varying regression models have been widely employed to study trends and evolving relationships in empirical research. We developed a Python package, **PyTimeVar**, which offers recently introduced nonparametric estimation and bootstrap-assisted inference methods. The package includes six advanced bootstrap techniques for constructing pointwise confidence intervals and simultaneous bands. Currently, no other software package provides equivalent functionality. In addition, **PyTimeVar** encompasses other popular parametric and nonparametric methods for modeling trends and time-varying relationships, including the boosted Hodrick-Prescott filter, power-law model, state-space model, and score-driven model. These methods have been specifically tailored for time-varying regression models in the package, enabling users to easily compare different approaches without the need to adapt general-purpose packages or switch to another programming language for additional methods.

Future work will focus on expanding the package to include additional models for trend analysis and time-varying relationships. The package is built for extensibility, allowing for easy integration of new features. Furthermore, extending the package to handle missing data would be a valuable enhancement.

Acknowledgments

We thank Siem Jan Koopman for his valuable suggestions during the preparation of this package. We also thank Henry de Vries, Ignace De Vos, Simeon Klumperbeek, and Chenhui Wang for testing the package.

References

Anand B, Paul S, Nair AR (2023). "Time-varying effects of oil price shocks on financial stress: Evidence from India." *Energy Economics*, p. 106703.

Ang A, Kristensen D (2012). "Testing conditional factor models." *Journal of Financial Economics*, **106**(1), 132–156.

Ardia D, Boudt K, Catania L (2019). "Generalized autoregressive score models in R: The GAS package." *Journal of Statistical Software*, **88**, 1–28.

Artemova M, Blasques F, van Brummelen J, Koopman SJ (2022a). "Score-driven models: Methodology and theory." In *Oxford Research Encyclopedia of Economics and Finance*. Oxford University Press.

Artemova M, Blasques F, van Brummelen J, Koopman SJ (2022b). "Score-driven models: Methods and applications." In *Oxford Research Encyclopedia of Economics and Finance*. Oxford University Press.

Awaworyi Churchill S, Inekwe J, Ivanovski K, Smyth R (2020). "The Environmental Kuznets Curve across Australian states and territories." *Energy Economics*, **90**, 104869. doi:https://doi.org/10.1016/j.eneco.2020.104869.

Beutner E, Lin Y, Lucas A (2023). "Consistency, distributional convergence, and optimality of score-driven filters." Tinbergen Institute Discussion Paper, 2023-051/III.

Biswas E, Sabzikar F, Phillips PCB (2024). "Boosting the HP filter for trending time series with long-range dependence." *Econometric Reviews*, **0**(0), 1–39. doi:10.1080/07474938.2024.2380704.

Blasques F, van Brummelen J, Koopman SJ, Lucas A (2022). "Maximum likelihood estimation for score-driven models." *Journal of Econometrics*, **227**(2), 325–346. doi:https://doi.org/10.1016/j.jeconom.2021.06.003.

Bühlmann P (1998). "Sieve bootstrap for smoothing in nonstationary time series." *Annals of Statistics*, **26**(1), 48 – 83.

Cai Z (2007). "Trending time-varying coefficient time series models with serially correlated errors." *Journal of Econometrics*, **136**(1), 163–188.

Cai Z, Li Q, Park JY (2009). "Functional-coefficient models for nonstationary time series data." *Journal of Econometrics*, **148**(2), 101–113.

Casas I, Fernandez-Casal R (2019). "tvReg: Time-varying coefficient linear regression for single and multi-equations in R." *Available at SSRN 3363526*.

Cavaliere G, Taylor AMR (2009). "Heteroskedastic time series with a unit root." *Econometric Theory*, **25**(5), 1228–1276.

Chu CK, Marron JS (1991). "Comparison of two bandwidth selectors with dependent errors." *Annals of Statistics*, **19**(4), 1906–1918.

Chudik A, Pesaran MH, Smith RP (2023). "Pooled Bewley estimator of long run relationships in dynamic heterogenous panels." *Econometrics and Statistics*.

Craven P, Wahba G (1978). "Smoothing noisy data with spline functions." *Numerische Mathematik*, **31**(4), 377–403. ISSN 0945-3245. doi:10.1007/BF01404567. URL https://doi.org/10.1007/BF01404567.

Creal D, Koopman SJ, Lucas A (2013). "Generalized autoregressive score models with applications." *Journal of Applied Econometrics*, **28**(5), 777–795. doi:https://doi.org/10.1002/jae.1279.

Davidson R, Flachaire E (2008). "The wild bootstrap, tamed at last." *Journal of Econometrics*, **146**(1), 162–169.

Ding X, Zhou Z (2023). "AutoRegressive approximations to nonstationary time series with inference and applications." *Annals of Statistics*, **51**(3), 1207–1231.

D'Innocenzo E, Luati A, Mazzocchi M (2023). "A robust score-driven filter for multivariate time series." *Econometric Reviews*, **42**(5), 441–470.

Duckworth D, Balatsko M (2024). *pykalman: An implementation of the Kalman Filter, Kalman Smoother, and EM algorithm in Python*. Python package version 0.9.7, URL https://pypi.org/project/pykalman/.

Durbin J, Koopman SJ (2012). *Time series analysis by state space methods*. 2nd edition. Oxford University Press (Oxford). doi:10.1093/acprof:oso/9780199641178.001.0001.

Dziak JJ, Coffman DL, Li R, Litson K, Chakraborti Y (2023). *tvem: Time-Varying Effect Models*. doi:10.32614/CRAN.package.tvem. R package version 1.4.1, URL https://CRAN.R-project.org/package=tvem.

Friedrich M, Beutner E, Reuvers H, Smeekes S, Urbain JP, Bader W, Franco B, Lejeune B, Mahieu E (2020a). "A statistical analysis of time trends in atmospheric ethane." *Climatic Change*, **162**, 105–125.

Friedrich M, Lin Y (2024). "Sieve bootstrap inference for linear time-varying coefficient models." *Journal of Econometrics*, **239**(1), 105345. doi:https://doi.org/10.1016/j.jeconom.2022.09.004.

Friedrich M, Smeekes S, Urbain JP (2020b). "Autoregressive wild bootstrap inference for nonparametric trends." *Journal of Econometrics*, **214**(1), 81–109.

Gao J, Peng B, Wu WB, Yan Y (2024). "Time-varying multivariate causal processes." *Journal of Econometrics*, **240**(1), 105671.

Gasperoni F, Luati A, Paci L, D'Innocenzo E (2023). "Score-driven modeling of spatio-temporal data." *Journal of the American Statistical Association*, **118**(542), 1066–1077.

Guo H, Wu C, Yu Y (2017). "Time-varying beta and the value premium." *Journal of Financial and Quantitative Analysis*, **52**(4), 1551–1576.

Harvey A, Trimbur T (2008). "Trend estimation and the Hodrick-Prescott filter." *Journal of the Japan Statistical Society*, **38**(1), 41–49.

Harvey AC (2013). *Dynamic models for volatility and heavy tails: with applications to financial and economic time series*, volume 52. Cambridge University Press.

Harvey AC (2022). "Score-driven time series models." *Annual Review of Statistics and Its Application*, **9**(1), 321–342.

Harvey AC, Luati A (2014). "Filtering with heavy tails." *Journal of the American Statistical Association*, **109**(507), 1112–1122.

Huang H, Leng X, Liu X, Peng L (2020). "Unified inference for an AR process regardless of finite or infinite variance GARCH errors." *Journal of Financial Econometrics*, **18**(2), 425–470.

Hubbard A (2024). *kalmanfilter: Kalman filter*. doi:10.32614/CRAN.package.kalmanfilter. R package version 2.1.1, URL https://CRAN.R-project.org/package=kalmanfilter.

Kapetanios G, Millard S, Petrova K, Price S (2020). "Time-varying cointegration with an application to the UK Great Ratios." *Economics Letters*, **193**, 109213. ISSN 0165-1765. doi:https://doi.org/10.1016/j.econlet.2020.109213.

Karmakar S, Richter S, Wu WB (2022). "Simultaneous inference for time-varying models." *Journal of Econometrics*, **227**(2), 408–428.

Koopman SJ, Lucas A, Scharth M (2016). "Predicting time-varying parameters with parameter-driven and observation-driven models." *Review of Economics and Statistics*, **98**(1), 97–110.

Kreiss JP, Paparoditis E, Politis DN (2011). "On the range of validity of the autoregressive sieve bootstrap." *Annals of Statistics*, **39**(4), 2103 – 2130. doi:10.1214/11-AOS900.

Kunsch HR (1989). "The jackknife and the bootstrap for general stationary observations." *Annals of Statistics*, pp. 1217–1241.

Li D, Phillips PC, Gao J (2020). "Kernel-based inference in time-varying coefficient cointegrating regression." *Journal of Econometrics*, **215**(2), 607–632.

Li Q, Racine JS (2007). *Nonparametric Econometrics: Theory and Practice*. Princeton University Press.

Li X, Zhao Z (2019). "A time varying approach to the stock return–inflation puzzle." *Journal of the Royal Statistical Society Series C: Applied Statistics*, **68**(5), 1509–1528.

Lin Y, Reuvers H (2024). "Cointegrating polynomial regressions with power law trends." Tinbergen Institute Discussion Paper, 2022-092/III.

Lin Y, Song M, van der Sluis B (2024). "Bootstrap inference for linear time-varying coefficient models in locally stationary time series." *Journal of Computational and Graphical Statistics*. doi:https://doi.org/10.1080/10618600.2024.2403705. Forthcoming.

Lin Y, van der Sluis B, Friedrich M (2023). "Bootstrapping trending time-varying coefficient panel models with missing observations." Tinbergen Institute Discussion Paper, 23-079/III.

Mei Z, Phillips PC, Shi Z (2024). "The boosted Hodrick-Prescott filter is more general than you might think." *Journal of Applied Econometrics*. doi:https://doi.org/10.1002/jae.3086.

Mudelsee M (2019). "Trend analysis of climate time series: A review of methods." *Earth-science reviews*, **190**, 310–322.

Ng E, Wang Z, Chen H, Yang S, Smyl S (2020). "Orbit: Probabilistic forecast with exponential smoothing." 2004.08492.

Nishi M (2024). "Estimating time-varying parameters of various smoothness in linear models via kernel regression." *arXiv preprint arXiv:2406.14046*.

Phillips PC, Li D, Gao J (2017). "Estimating smooth structural change in cointegration models." *Journal of Econometrics*, **196**(1), 180–195.

Phillips PC, Shi Z (2021). "Boosting: Why you can use the HP filter." *International Economic Review*, **62**(2), 521–570.

Ren B, Lucey B (2023). "Herding in the Chinese renewable energy market: Evidence from a bootstrapping time-varying coefficient autoregressive model." *Energy Economics*, **119**, 106526.

Robinson PM (2012). "Inference on Power Law Spatial Trends." *Bernoulli*, **18**, 644–677.

Seabold S, Perktold J (2010). "Statsmodels: Econometric and statistical modeling with Python." *SciPy*, **7**(1).

Smeekes S, Taylor AMR (2012). "Bootstrap union tests for unit roots in the presence of nonstationary volatility." *Econometric Theory*, **28**(2), 422–456. doi:10.1017/S0266466611000387.

Smeekes S, Urbain JP (2014). "A multivariate invariance principle for modified wild bootstrap methods with an application to unit root testing." doi:https://doi.org/10.26481/umagsb.2014008. Maastricht University Research Papers, RM/14/008.

Smeekes S, Wilms I (2023). "bootUR: An R package for bootstrap unit root tests." *Journal of Statistical Software*, **106**, 1–39.

Uddin MM, Mishra V, Smyth R (2020). "Income inequality and CO2 emissions in the G7, 1870-2014: Evidence from non-parametric modelling." *Energy Economics*, **88**, 104780. ISSN 0140-9883. doi:https://doi.org/10.1016/j.eneco.2020.104780.

Umlandt D (2023). "Score-driven asset pricing: Predicting time-varying risk premia based on cross-sectional model performance." *Journal of Econometrics*, **237**(2), 105470.

Vieu P (1991). "Nonparametric Regression: Optimal Local Bandwidth Choice." *Journal of the Royal Statistical Society: Series B (Methodological)*, **53**(2), 453–464. doi:https://doi.org/10.1111/j.2517-6161.1991.tb01837.x.

Wales DJ, Doye JP (1997). "Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms." *Journal of Physical Chemistry A*, **101**(28), 5111–5116.

Wu W, Zhou Z (2017). "Nonparametric inference for time-varying coefficient quantile regression." *Journal of Business & Economic Statistics*, **35**(1), 98–109.

Zhou Z, Wu WB (2010). "Simultaneous Inference of Linear Models with Time Varying Coefficients." *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, **72**, 513–531.

Zou X, Lin Y, Lucas A (2024). "Closing the gap between state-space and score-driven models: An application to modeling implied volatility surface dynamics." Working Paper.

A. Additional algorithms

A.1. Pointwise confidence intervals and simultaneous confidence bands

Algorithm 3: $(1-\alpha)$-level pointwise intervals and simultaneous bands

1 **if** *Pointwise confidence intervals* **then**
2 **for** $j \in \{0, \ldots, d\}$ **do**
3 **for** $\tau \in (0,1)$ **do**
4 compute $I^*_{j,\alpha}(\tau) = \left[\hat{\beta}_j(\tau) - \hat{q}_{j,1-\alpha/2}(\tau),\ \hat{\beta}_j(\tau) - \hat{q}_{j,\alpha/2}(\tau)\right]$;
5 **end**
6 **end**
7 **else if** *Simultaneous confidence bands* **then**
8 **for** $\tau_i \in (0,1)$ **do**
9 Obtain $U_i(h) = [\tau_i - ah,\ \tau_i + bh] \bigcap [0,1]$, $G = \bigcup_{i=1}^{m} U_i(h)$;
10 **end**
11 **for** $j \in \{0, \ldots, d\}$ **do**
12 **for** $\alpha_p \in [1/B, \alpha]$ **do**
13 **for** $\tau \in G$ **do**
14 Compute $\hat{q}_{j,1-\alpha_p/2}(\tau)$ and $\hat{q}_{j,\alpha_p/2}(\tau)$;
15 **end**
16 **end**
17 Compute $\hat{\alpha}_s = \text{argmin}_{\alpha_p \in [1/B,\alpha]} \Big| \mathbb{P}^*(\hat{q}_{j,\alpha_p/2}(\tau) \leq \hat{\beta}^*_j(\tau) - \tilde{\beta}_j(\tau) \leq \hat{q}_{j,1-\alpha_p/2}(\tau),$ $\forall \tau \in G) - (1-\alpha) \Big|$;
18 **for** $\tau \in G$ **do**
19 Compute $I^{G*}_{j,\hat{\alpha}_s}(\tau) = \left[\hat{\beta}_j(\tau) - \hat{q}_{j,1-\hat{\alpha}_s/2}(\tau),\ \hat{\beta}_j(\tau) - \hat{q}_{j,\hat{\alpha}_s/2}(\tau)\right]$.
20 **end**
21 **end**
22 **end**

Line 17 in Algorithm 3 states that $\hat{\alpha}_s$ is determined based on:

$$\frac{\#\left\{\hat{\beta}^*_j(\tau) - \tilde{\beta}_j(\tau) \in \left[\hat{q}_{j,\hat{\alpha}_s/2}(\tau), \hat{q}_{j,1-\hat{\alpha}_s/2}(\tau)\right],\ \forall \tau \in G\right\}}{B} \approx 1-\alpha,$$

where $\#E$ counts the number of times the event E appears in the bootstrap samples.

A.2. Algorithm for boosted HP filter

Algorithm 4 outlines the steps for implementing the boosted HP filter.

Algorithm 4: Boosted HP filter

Set $m = 1$ and compute HP filter estimates $\{\hat{\beta}_{0,t}^{(m)}\}$ from $\{y_t\}$;
repeat
  Compute residuals $\hat{u}_t^{(m)} = y_t - \hat{\beta}_{0,t}^{(m)}$, $t = 1, \ldots, n$;
  Fit HP filter to residuals $\{\hat{u}_t^{(m)}\}$, resulting in estimates $\{\tilde{\beta}_{0,t}^{(m+1)}\}$;
  Update $\hat{\beta}_{0,t}^{(m+1)} = \hat{\beta}_{0,t}^{(m)} + \tilde{\beta}_{0,t}^{(m+1)}$ and set $m = m + 1$;
until *Stopping criterion is met*;

A.3. Algorithm for state-space models

Table 5: Dimensions of parameters in state-space models and Kalman filter and smoother

State-space models			Kalman filter and smoother		
scalar	$(d+1) \times 1$	$(d+1) \times (d+1)$	scalar	$(d+1) \times 1$	$(d+1) \times (d+1)$
y_t	$\boldsymbol{x}_t$	$\boldsymbol{T}$	v_t	$\boldsymbol{b}_t$	$\boldsymbol{P}_t$
u_t	$\boldsymbol{\beta}_t$	$\boldsymbol{R}$	f_t	$\boldsymbol{k}_t$	$\boldsymbol{V}_t$
σ_u	$\boldsymbol{\eta}_t$	$\boldsymbol{Q}$		$\boldsymbol{r}_t$	$\boldsymbol{N}_t$
				$\boldsymbol{b}_{t\|t}$	$\boldsymbol{P}_{t\|t}$
				$\hat{\boldsymbol{\beta}}_t$	

For a clear overview, the dimensions of the quantities in the state-space model, as well as those used in the Kalman filter and smoother, are summarized in Table 5. The algorithms for implementing the Kalman filter and smoother are provided in Algorithms 5 and 6 below.

Algorithm 5: Kalman filter

Initialize $\boldsymbol{b}_1$, $\boldsymbol{P}_1$;
for $t \in \{1, \ldots, n\}$ **do**
  $v_t = y_t - \boldsymbol{x}_t^\top \boldsymbol{b}_t$;
  $f_t = \boldsymbol{x}_t^\top \boldsymbol{P}_t \boldsymbol{x}_t + \sigma_u$;
  $\boldsymbol{k}_t = \boldsymbol{T}\boldsymbol{P}_t \boldsymbol{x}_t f_t^{-1}$;
  $\boldsymbol{b}_{t|t} = \boldsymbol{b}_t + \boldsymbol{P}_t \boldsymbol{x}_t f_t^{-1} v_t$;
  $\boldsymbol{b}_{t+1} = \boldsymbol{T}\boldsymbol{b}_{t|t}$;
  $\boldsymbol{P}_{t+1} = \boldsymbol{T}\boldsymbol{P}_t\boldsymbol{T}^\top + \boldsymbol{R}\boldsymbol{Q}\boldsymbol{R}^\top - \boldsymbol{k}_t f_t \boldsymbol{k}_t^\top$;
end
return filtered states
$\boldsymbol{b}_{\text{filt}} = (\boldsymbol{b}_{1|1}, \boldsymbol{b}_{2|2}, \ldots, \boldsymbol{b}_{n|n})$ and
predicted states
$\boldsymbol{b}_{\text{pred}} = (\boldsymbol{b}_2, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{n+1})$.

Algorithm 6: Kalman smoother

Set $\boldsymbol{r}_n = \boldsymbol{0}$, $\boldsymbol{N}_n = \boldsymbol{0}$;
for $t \in \{n, n-1, \ldots, 1\}$ **do**
  $\boldsymbol{L}_t = \boldsymbol{T} - \boldsymbol{k}_t \boldsymbol{x}_t^\top$;
  $\boldsymbol{r}_{t-1} = \boldsymbol{x}_t f_t^{-1} v_t + \boldsymbol{L}_t^\top \boldsymbol{r}_t$;
  $\boldsymbol{N}_{t-1} = \boldsymbol{x}_t f_t^{-1} \boldsymbol{x}_t^\top + \boldsymbol{L}_t^\top \boldsymbol{N}_t \boldsymbol{L}_t$;
  $\hat{\boldsymbol{b}}_t = \boldsymbol{b}_t + \boldsymbol{P}_t \boldsymbol{r}_{t-1}$;
  $\boldsymbol{V}_t = \boldsymbol{P}_t - \boldsymbol{P}_t \boldsymbol{N}_{t-1} \boldsymbol{P}_t$;
end
return smoothed states
$\hat{\boldsymbol{b}} = (\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \ldots, \hat{\boldsymbol{b}}_T)$.

A.4. Algorithm for score-driven models

Algorithm 7: Score-driven model

1 Set initial values $\boldsymbol{\beta}_0$ using 10% of the data, define lower bounds $\boldsymbol{\gamma}_L$ and upper bounds $\boldsymbol{\gamma}_U$ for $\boldsymbol{\gamma}$;
2 **if** *t-GAS* **then**
3 Extend bounds $\boldsymbol{\gamma}_L$ and $\boldsymbol{\gamma}_U$ to account for t-distribution parameters;
4 **end**
5 **if** *$\mathcal{N}$-GAS* **then**
6 Maximize $\mathcal{L}(\boldsymbol{\gamma}; \boldsymbol{\beta}_0)$ with respect to $\boldsymbol{\gamma}$ and obtain the maximizer $\hat{\boldsymbol{\gamma}} = (\hat{\sigma}_u, \hat{\boldsymbol{\omega}}^\top, \hat{\boldsymbol{B}}, \hat{\boldsymbol{A}})^\top$, where $\mathcal{L}$ is constructed as in Algorithm 8;
7 Set $\hat{\boldsymbol{\beta}}_1 = \boldsymbol{\beta}_0$;
8 **for** $t \in \{1, \ldots, n\}$ **do**
9 Compute the scaled density score $\hat{\boldsymbol{s}}_t = \boldsymbol{x}_t(y_t - \hat{\boldsymbol{\beta}}_t^\top \boldsymbol{x}_t)$;
10 Update $\hat{\boldsymbol{\beta}}_{t+1} = \hat{\boldsymbol{\omega}} + \hat{\boldsymbol{B}}\hat{\boldsymbol{\beta}}_t + \hat{\boldsymbol{A}}\hat{\boldsymbol{s}}_t$;
11 **end**
12 **else if** *t-GAS* **then**
13 Maximize $\mathcal{L}(\boldsymbol{\gamma}; \boldsymbol{\beta}_0)$ with respect to $\boldsymbol{\gamma}$ and obtain the maximizer $\hat{\boldsymbol{\gamma}} = (\hat{\nu}, \hat{\sigma}_u, \hat{\boldsymbol{\omega}}^\top, \hat{\boldsymbol{B}}, \hat{\boldsymbol{A}})^\top$, where $\mathcal{L}$ is constructed as in Algorithm 8;
14 Follow Lines 7 - 11 except the scaled density score in Line 9 should be replaced by the following: $\hat{\boldsymbol{s}}_t = (1+\nu)^{-1}(3+\nu)(1+\nu^{-1})\left[1 + \nu^{-1}\left(\frac{y_t - \hat{\boldsymbol{\beta}}_t^\top \boldsymbol{x}_t}{\sigma_u}\right)^2\right]^{-1} \boldsymbol{x}_t(y_t - \hat{\boldsymbol{\beta}}_t^\top \boldsymbol{x}_t)$.
15 **end**

We provide further details on implementing score-driven models. Algorithm 7 outlines the steps for applying both $\mathcal{N}$-GAS and t-GAS models. For $\mathcal{N}$-GAS models, assuming $u_t \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_u^2)$ in Eq. (1), we have $\boldsymbol{\nabla}_t = \sigma_u^{-2}\boldsymbol{x}_t(y_t - \boldsymbol{\beta}_t^\top \boldsymbol{x}_t)$, and set $\boldsymbol{S}_t = \sigma_u^2$, leading to $\hat{\boldsymbol{s}}_t = \boldsymbol{S}_t\boldsymbol{\nabla}_t = \boldsymbol{x}_t(y_t - \hat{\boldsymbol{\beta}}_t^\top \boldsymbol{x}_t)$ in Line 9 (Algorithm 7). Similarly, for t-GAS, assuming $u_t \overset{\text{i.i.d.}}{\sim} t(0, \sigma_u^2, \nu)$ in Eq. (1), we have $\boldsymbol{\nabla}_t = (1+\nu^{-1})\left[1 + \nu^{-1}\left(\frac{y_t - \boldsymbol{\beta}_t^\top \boldsymbol{x}_t}{\sigma_u}\right)^2\right]^{-1} \sigma_u^{-2}\boldsymbol{x}_t(y_t - \boldsymbol{\beta}_t^\top \boldsymbol{x}_t)$. By setting $\boldsymbol{S}_t = (1+\nu)^{-1}(3+\nu)\sigma_u^2$, we obtain $\hat{\boldsymbol{s}}_t$ in Line 14.

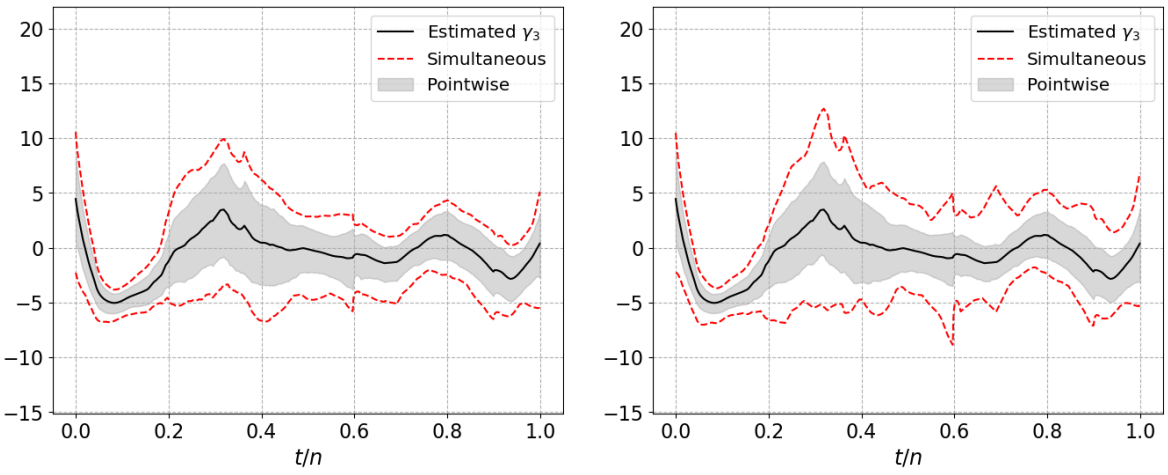
Moreover, Algorithm 8 constructs the log-likelihood function required in Algorithm 7. For $\mathcal{N}$-GAS, we have $\log p(y_t \mid \boldsymbol{x}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma}) \propto -\log \sigma_u - \frac{1}{2}\left(\frac{y_t - \boldsymbol{\beta}_t^\top \boldsymbol{x}_t}{\sigma_u}\right)^2$, and for t-GAS, $\log p(y_t \mid \boldsymbol{x}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma}) = \log \frac{\Gamma\left((\nu+1)/2\right)}{\sqrt{\pi\nu}\,\Gamma\left(\nu/2\right)} - \log \sigma_u - \frac{\nu+1}{2}\log\left[1 + \nu^{-1}\left(\frac{y_t - \boldsymbol{\beta}_t^\top \boldsymbol{x}_t}{\sigma_u}\right)^2\right]$.

Algorithm 8: Construct likelihood (lh) function

Set $\boldsymbol{\beta}_{\text{current}} = \boldsymbol{\beta}_0$ and $\text{lh} = 0$;
if $\mathcal{N}$-*GAS* **then**
  for $t \in \{1 \dots n\}$ **do**
    Update $\text{lh} = \text{lh} + \left(\frac{y_t - \boldsymbol{\beta}_{\text{current}}^\top \boldsymbol{x}_t}{\sigma_u} \right)^2$;
    Compute $\boldsymbol{s}_t = \boldsymbol{x}_t (y_t - \boldsymbol{\beta}_{\text{current}}^\top \boldsymbol{x}_t)$;
    Update $\boldsymbol{\beta}_{\text{current}} = \boldsymbol{\omega} + \boldsymbol{B}\boldsymbol{\beta}_{\text{current}} + \boldsymbol{A}\boldsymbol{s}_t$;
  end
  $\text{lh} = -\log \sigma_u - 0.5 \cdot \text{lh}/n$;
else if t-*GAS* **then**
  for $t \in \{1 \dots n\}$ **do**
    Update $\text{lh} = \text{lh} + \left(\frac{y_t - \boldsymbol{\beta}_{\text{current}}^\top \boldsymbol{x}_t}{\sigma_u} \right)^2$;
    Compute

$$\boldsymbol{s}_t = (1+\nu)^{-1}(3+\nu)(1+\nu^{-1}) \left[1 + \nu^{-1} \left(\frac{y_t - \boldsymbol{\beta}_{\text{current}}^\top \boldsymbol{x}_t}{\sigma_u} \right)^2 \right]^{-1} \boldsymbol{x}_t (y_t - \boldsymbol{\beta}_{\text{current}}^\top \boldsymbol{x}_t);$$

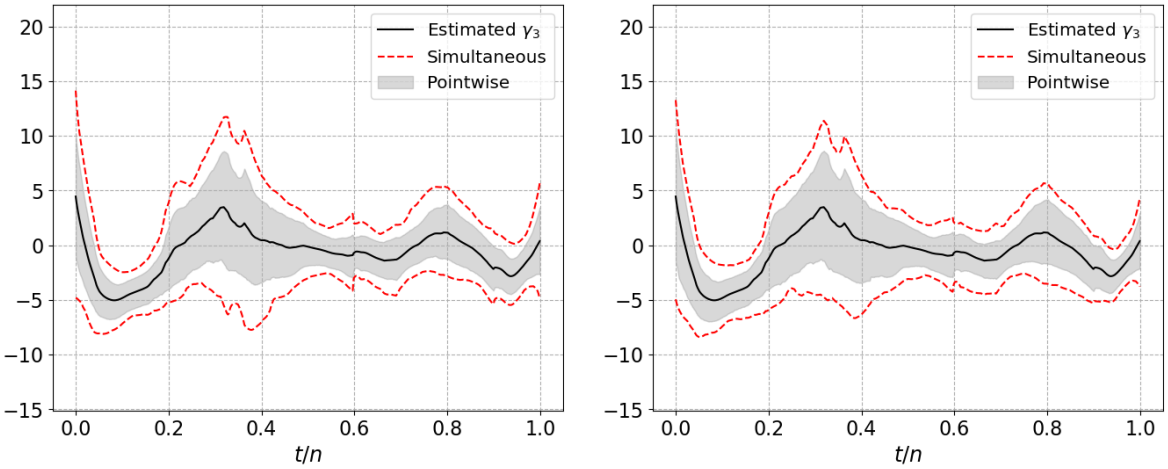
    Update $\boldsymbol{\beta}_{\text{current}} = \boldsymbol{\omega} + \boldsymbol{B}\boldsymbol{\beta}_{\text{current}} + \boldsymbol{A}\boldsymbol{s}_t$;
  end
  $\text{lh} = -0.5(\nu+1) \cdot \text{lh}/n + \log \Gamma((\nu+1)/2) - \log \Gamma(\nu/2) - 0.5 \log(\pi\nu) - \log \sigma_u$;
end
return lh.



(a) SB (b) SWB

Figure 9: 95%-level pointwise intervals (shaded) and simultaneous bands (red dashed lines) using different bootstrap methods for the herding effects example in Section 4.2

A.5. Additional empirical results



(c) WB (d) AWB

Figure 9: Continuation of Figure 9